Rodrigo
Lopes da Cunha

Determinação de tráfego para uplink de vídeo e otimização de rede

Uplink video traffic determination and network optimization

Universidade de Aveiro
2017

Departamento de Eletrónica,
Telecomunicações e Informática

Rodrigo
Lopes da Cunha

Determinação de tráfego para uplink de vídeo e otimização de rede

Uplink video traffic determination and network optimization

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica do Doutor Rui Luís Andrade Aguiar, Professor catedrático do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro, e do Doutor Paulo Jorge Salvador Serra Ferreira, Professor auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro.

o júri / the jury

presidente / president

Prof. Doutor André Ventura da Cruz Marnoto Zúquete
Professor auxiliar da Universidade de Aveiro

vogais / examiners committee

Prof. Doutor Rui Luís Andrade Aguiar
Professor catedrático da Universidade de Aveiro

Prof. Doutor Paulo Manuel Martins de Carvalho
Professor associado da Universidade do Minho

agradecimentos /
acknowledgements

Gostaria inicialmente de agradecer ao Prof. Doutor Rui Aguiar (Orientador), Prof. Doutor Paulo Salvador (Co-Orientador) e Prof. Doutor Daniel Corujo, pelo apoio e acompanhamento dado ao longo da elaboração desta dissertação.

Agradeço ainda ao Doutor Sérgio Figueiredo (Altran) pela oportunidade, bem como pela ajuda dada ao longo do projeto em que esta disertação esteve envolvida.

Finalmente, agradeço aos meus pais, ao meu irmão, à minha namorada e aos meus amigos, que ao longo deste tempo sempre me apoiaram e motivaram e sem eles nada disto seria possível.

Palavras Chave

Monitorização, Redes Definidas por Software, Caracterização de Tráfego, Análise de Pacotes de Rede, Qualidade de Serviço

Resumo

Com o aumento do número de plataformas de transmissão de vídeo, as operadoras têm sofrido uma maior sobrecarga nas suas redes. De forma a fornecer uma melhor gestão dessas mesmas redes, garantindo qualidade de serviço a todos os clientes, torna-se necessário dar prioridade ao tráfego correspondente a vídeo aplicando novos conveitos na área das telecomunicações, como é o caso de Software-Defined Networking. Esta dissertação procura, numa primeira fase, apresentar uma revisão de vários temas relacionados com a determinação de tráfego de vídeo, Software-Defined Networking e qualidade de serviço. Posteriormente, é apresentada uma solução de uma aplicação de monitorização, que tem como objetivo, a deteção de tráfego de vídeo, de forma a ajudar na priorização de tráfego e na otimização da rede. A solução é validada através de uma implementação, baseada na performance e na baixa latência do sistema, que procura responder o mais rápido possível com informação sobre um determinado fluxo de pacotes na rede. São ainda apresentados resultados relativos a esta implementação.

Keywords

Monitoring, Software-Defined Networking, Traffic Characterization, Network Packets Analysis, Quality of Service

Abstract

With the increase of live streaming platforms, service providers have been experiencing a overhead on their networks. In order to provide a better management of these networks, ensuring quality of service to all customers, it is necessary to prioritize video traffic using new concepts being introduced into the telecommunications field, such as Software-Defined Networking. Firstly, this dissertation aims to present a review of several topics related with video traffic determination, Software-Defned Networking and quality of service. Secondly, a monitoring application solution is presented, which aims to detect video traffic in order to help the prioritization of traffic and network optimization. The solution is validated through an implementation, based on the system's performance and low latency, which tries to reply as quickly as possible with information about a certain flow of network packets. Results related with this implementation are also presented.

Contents

List of Figures

List of Tables

Acronyms

ABR	Adaptive Bitrate Streaming
AP	Access Point
API	Application Programming Interface
APN	Access Point Name
APN-AMBR	Access Point Name Aggregate Maximum Bitrate
AWS	Amazon Web Services
BS	Base Station
CFP	Contention Free Period
CLI	Command-line Interface
CP	Contention Period
CSMA/CA	Carrier Sense Multiple Access – Collision Avoidance
CTS	Clear to Send
CWT	Continuous Wavelet Transform
DASH	Dynamic Adaptive Streaming over HTTP
DCF	Distributed Coordination Function
EDCA	Enhanced Distributed Channel Access
EPS	Evolved Packet System
GBR	Guaranteed Bit Rate
HCCA	HCF Controller Channel Access
HCF	Hybrid Coordination Function
HLS	HTTP Live Streaming
HTTP	Hypertext Transfer Protocol
ICE	Interactive Connectivity Establishment
IP	Internet Protocol
ISP	Internet Service Provider
IT	Institute of Telecommunications
LFI	Link Fragmentation and Interleaving
LTE	Long-Term Evolution
MBR	Maximum Bit Rate
MME	Mobility Management Entity
MPD	Media Presentation Description
MSS	Microsoft Smooth Streaming
NAT	Network Address Translation
ODL	Open Daylight
ONF	Open Networking Foundation
ONOS	Open Network Operating System
OVS	Open vSwitch
P-GW	Packet Data Network Gateway
PCF	Point Coordination Function
PoC	Proof of Concept
QCI	QoS Class Identifier
QoS	Quality of Service
RED	Random Early Detection
RTCP	Real-Time Control Protocol
RTMFP	Real-Time Media Flow Protocol
RTMP	Real-Time Messaging Protocol
RTP	Real-Time Transport Protocol
RTS	Request to Send
RTSP	Real-Time Streaming Protocol
SDN	Software-Defined Networking
SLA	Service Layer Abstraction
SSL	Secure Sockets Layer
STUN	Session Traversal Utilities for NAT

TC	Traffic Categories
TCP	Transmission Control Protocol
TLS	Transport Layer Security
TOS	Type of Service
TURN	Traversal Using Relays around NAT
UDP	User Datagram Protocol
UE	User Equipment
UE-AMBR	User Equipment Aggregate Maximum Bitrate
VM	Virtual Machine
VSA	Video SDN Application
WAN	Wide Area Network
WLAN	Wireless Local Area Network
WebRTC	Web Real-Time Communication

CHAPTER 1

Introduction

In today's network architectures, the most common direction of information flows, downlink, is favored. This process allows greater bandwidth during a download in comparison with an upload, reflecting the consumerist nature of Internet access. Nowadays, the production of content supported by video live streaming services, such as Facebook Live[1], Twitch[2], Twitter's Periscope[3], the evolution of mobile phones' multimedia capabilities and the possibility of connecting to different types of access networks, are changing the current trend.

Since that it has become normal to do a video live stream during an event, such as concerts, conferences, football games, among others, the network has been facing a problem related to the amount of traffic generated by each user. For example, during a large event with thousands of attendees, the amount of generated traffic reaches high values and can congest the network, which prevents users from broadcasting to their friends what they are watching.

According to [1], during a normal day, the ratio of downlink to uplink traffic is in order of 9:1, but it changes in large events and descends to approximately 1:1, which constitutes a problem to the cellular service provider.

The combination of a large number of users that are only consumers, with a quickly growth of information producers, creates a big challenge to the telecommunications operators regarding the network control, management and optimization. Additionally, with the increase of mobile phones subscribers and users of the mentioned video services

[1] `https://live.fb.com/`
[2] `https://www.twitch.tv/`
[3] `https://www.periscope.tv/`

it becomes necessary to develop new mechanisms to face the growth of network video flows.

Initially, such mechanisms start with the determination and identification of the video flows produced by the users, distinguishing them from the others, such as Web traffic and Email. After this identification, it is possible to activate processes in the network, using Software-Defined Networking (SDN) concepts, to handle video traffic differently from other content. An example of such a process is the prioritization of video flows in the operator's network.

This dissertation aims to create the necessary mechanisms to detect video flows and differentiate them from other types of traffic. For this, different algorithms capable of optimally responding to each request by the user will be studied and investigated.

1.1 MOTIVATION

VDSNet is a project funded by Altran[4], an international group which offers innovation and high-tech engineering consulting, and developed in a partnership between Altran Research and Institute of Telecommunications (IT). It aims to provide a solution to telecommunication operators that focuses on the identification of the benefits of the integration of SDN technologies into scenarios where a user generates video with his/her mobile phone or a similar gadget. Additionally, the project aims to research how the SDN paradigm can be adapted to the specificities of live video streaming operation.

Moreover, SDN is a recent approach which is growing rapidly and it is likely that network operators will change their current solutions by SDN solutions. As explained in [2], SDN is seen as a way to simplify networking, and gives network operators more flexible and ease in QoS policies management.

The main motivation for this dissertation is the development of an algorithm for video detection in the network and its inclusion in a SDN solution.

1.2 OBJECTIVES

The VDSNet Project focuses on the optimization of uplink bandwidth and the prioritization of traffic when a video flow is detected. Thus, this dissertation is centered in the development of a solution (Monitor App) which is composed by a set of mechanisms

[4] `http://www.altran.com/`

that detect video flows in the network, and helps the process of QoS management during a live stream, optimizing it.

There are some objectives to be reached at the end of this dissertation, with the main focus being the study of different manners of video detection which will be done by mechanisms that analyze the Internet Protocol (IP), network ports, hostnames and periodicity of packets. This set of mechanisms working together forms the Monitor App and it is expected that it be able to notify other entities when a video flow is detected. Also, the time for the analysis of a flow should be the minimum possible, in order to quickly give feedback to other components of the final solution. The Monitor App can be used by multiple systems, but in this case it is part of a Proof of Concept (PoC) project, VDSNet, therefore it will be showed in some demonstrations with different types of audiences.

Additionally, this document aims to provide a description of all steps and analysis done during the development of the solution.

1.3 CONTRIBUTIONS

The work in this dissertation mainly contributes with a monitoring application that is contained in the SDN approach where the VDSNet Project fits, in order to separate monitoring from QoS and networking management. It will also contribute towards a PoC, that aims to show how the QoS policies can be used on a SDN architecture, when it is intended that only the video flows be prioritized. It can also be used by other systems, that wish to have access to the informations given by the Monitor App. Furthermore, it will be used in some demonstrations, such as schools, Altran, Students@deti and Teachers@deti, which contributes to prove its main objective and the dissemination of its benefits by an actual audience.

1.4 STRUCTURE

This document is split into six chapters, knowing that the first one (Introduction) was already presented, the remaining are organized as follows.

- **Chapter 2:** presentation of the state of the art. In this chapter the key concepts for the theme, will be presented. These will focus on multimedia protocols, video streaming technologies and SDN concepts.

- **Chapter 3:** presentation of all analysis which led to the solution proposed. An explained architecture is also presented in this chapter.
- **Chapter 4:** description of the implementation of the Monitor App and its mechanisms, that was performed for this dissertation. The architecture proposed in Chapter 3 is used to provide a detailed explanation of the implementation decisions that were taken.
- **Chapter 5:** presentation and analysis of the results obtained from testing the implemented solution. In this chapter all the tests are presented and the results are analyzed under a feasibility perspective, in order to verify the accomplishment of the presented objectives.
- **Chapter 6:** presentation of an overview of all work done during the development of the solution realized in this dissertation. Additionally, possible future work is also presented.

CHAPTER 2

State of the Art

2.1 video streaming technologies

As described in Chapter 1, this dissertation will focus on streaming services, which implies the study of associated technologies and/or protocols. There are many different protocols to do video streaming, each one with its own specifications, which will be analyzed in this section.

2.1.1 real-time streaming protocol

According to [3] and [4], Real-Time Streaming Protocol (RTSP) is an application-level protocol that establishes and controls streams of continuous media. Typically based on the Transmission Control Protocol (TCP) for reliable delivery, it has a very similar operation and syntax to Hypertext Transfer Protocol (HTTP) and is used by the client application to communicate with the server information aspects such as, the media file being requested, the type of application the client is using, the mechanism of delivery (User Datagram Protocol (UDP) or TCP) and other important control information commands such as DESCRIBE, SETUP and PLAY. Every session is maintained by the server using an identifier, being RTSP a stateful protocol.

There are two protocols, Real-Time Transport Protocol (RTP) and Real-Time Control Protocol (RTCP) which working together compose the RTSP protocol to deliver content to the user.

The first (RTP) is used for the transport of real-time multimedia (audio and video) and uses the TCP protocol in environments that suffer high packet loss or the UDP

protocol to solve the delay sensitive of the delivery. The RTP flow is unidirectional from the server to the client and the source port used by the server is always even.

The second one (RTCP) is a complementary protocol to RTP and is a bidirectional UDP-based mechanism to allow the client to communicate information about the stream quality to the server. The RTCP communication always uses the next UDP source port above the one used by the RTP stream, and consequently, always odd.

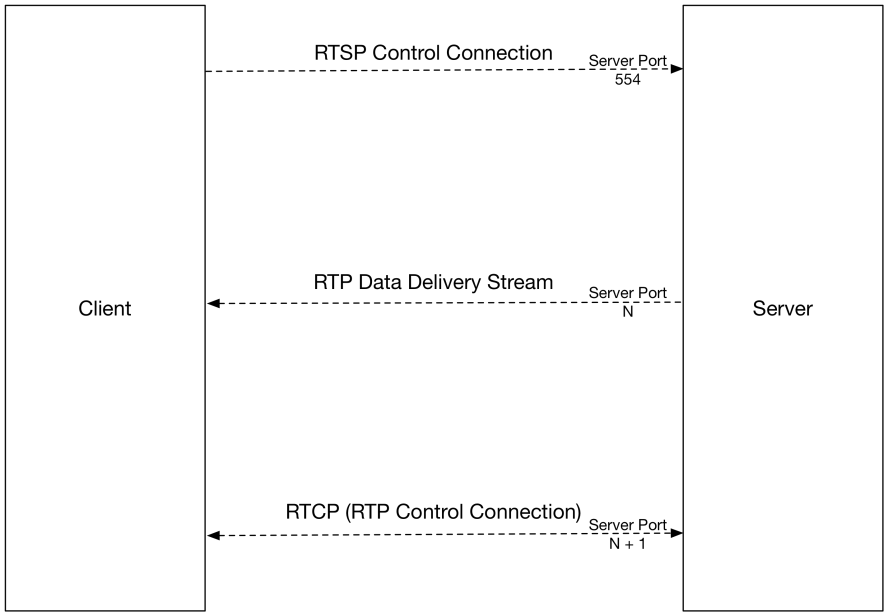

Figure 2.1: Three main application protocols used in real-time streaming [3]

As shown in Figure 2.1, the client establishes a TCP connection to the sever on the well-known port for RTSP, 554, and starts issuing a series of RTSP header commands to describe to the server details of the session requirements, such as the version of RTSP it supports. The headers DESCRIBE and SETUP are used to pass this information. After the negotiation, the client will issue a PLAY command and instruct the server to start the delivery of the RTP data stream. During the stream, a bidirectional RTCP stream carries information about the quality of the RTP stream back to the server. The command TEARDOWN is issued to notify the server to cease the RTP delivery, and thereby close the stream.

2.1.2 REAL-TIME MESSAGING PROTOCOL

As stated in the Real-Time Messaging Protocol (RTMP) specification [5], RTMP is an application-level protocol developed by Macromedia (owned by Adobe[1]) and

[1] `http://www.adobe.com/`

designed for multiplexing and packetizing multimedia transport streams over TCP. The main use of it is to avoid latency in communication, deliver audio or video streams smoothly, by splitting them in fragments.

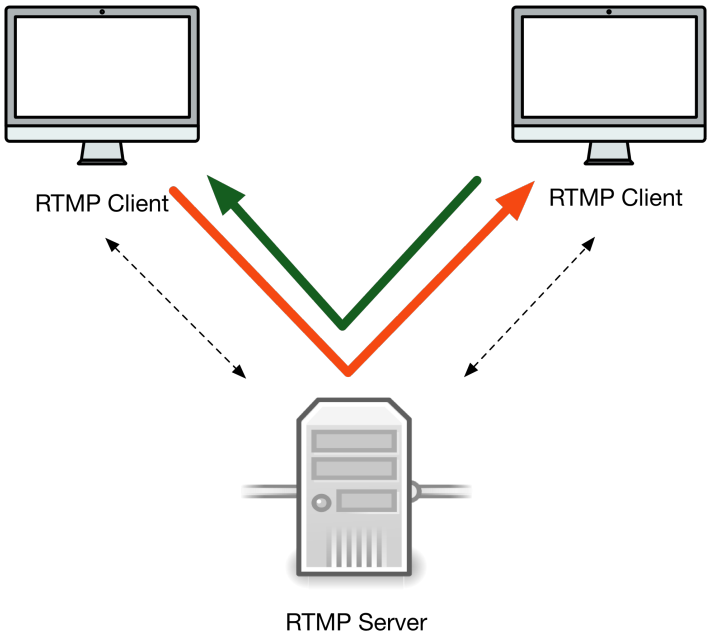

Figure 2.2: Communication using RTMP [6]

Initially, a connection is established with the server and during the communication, streams of video, audio and data messages, with the associated time information are sent through it using the TCP port number 1935 (Figure 2.2).

RTMP has also multiple variations, like, "plain" RTMP protocol (without any security), RTMPS (RTMP over Secure Sockets Layer (SSL)/Transport Layer Security (TLS)), RTMPE (RTMP encrypted using Adobe's own security mechanism) and RTMPT (encapsulated within HTTP requests to traverse firewalls).

There are multiple projects and platforms using RTMP, namely, Facebook[2], YouTube[3], Wowza[4], Flash Media Server[5], LiveCycle DS[6], among others.

2.1.3 REAL-TIME MEDIA FLOW PROTOCOL

Real-Time Media Flow Protocol (RTMFP) is a communication protocol from Adobe for multimedia delivery through client-server (RTMP) and peer-to-peer model.

[2] http://www.facebook.com/
[3] http://www.youtube.com/
[4] http://www.wowzamedia.com/
[5] http://www.adobe.com/products/flashmediaserver/
[6] http://www.adobe.com/products/livecycle/dataservices/

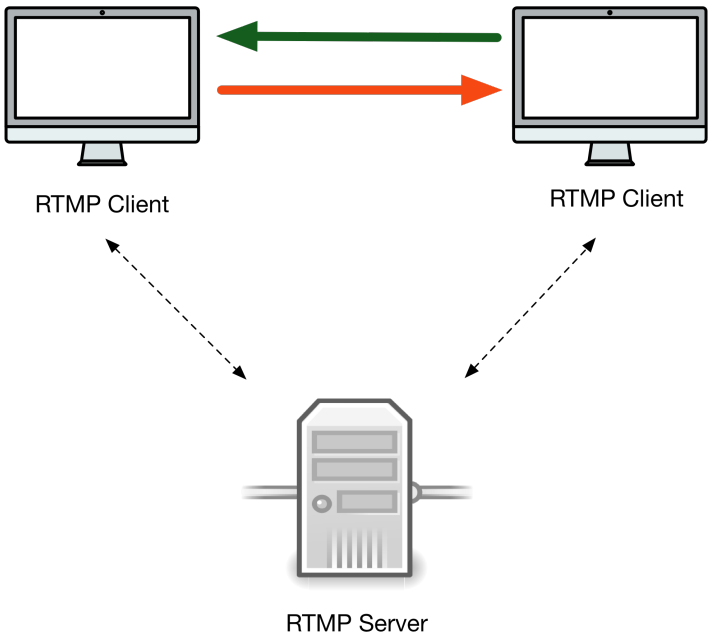

Figure 2.3: Communication using RTMFP [6]

Instead of connecting to the server to pass the data through it, like in RTMP, a connection identification is maintained over UDP with it and the clients connect directly to pass the data using UDP (Figure 2.3).

Comparatively to RTMP, this protocol uses less bandwidth and uses UDP instead of TCP, which is more efficient, with a lower latency and a greater tolerance for dropped or missing packets.

2.1.4 ADAPTIVE BITRATE STREAMING

Adaptive Bitrate Streaming (ABR) is a technology designed to deliver video to the user with the highest quality according the used device and it is an improvement of progressive video streaming, which consists in a single video file being streamed over the Internet [7].

As demonstrated in Figure 2.4, progressive streaming has some quality problems, as the picture will be stretched and it is possible to see pixelation when the screen resolution is higher than the video resolution. Buffering is also a problem, because if a consumer cannot download the video stream quickly enough it will need to pause, waiting for more data. These problems give users a poor quality of service.

ABR (Figure 2.5) solves all the problems of progressive streaming. To ensure

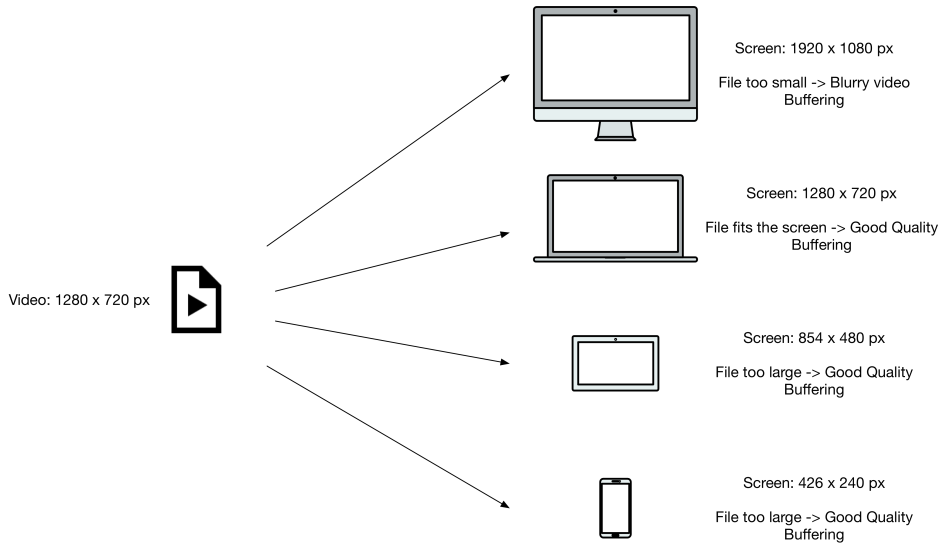

Figure 2.4: Progressive Streaming [7]

this, the source video is encoded into discrete file segments designated as "fragments" or "chunks". Each segment contains video data, audio data or other data with, for example, 2 seconds duration, is hosted on a HTTP server, which serves all clients, and is downloaded by them as a series of progressive downloads. According to [8], a sequence of fragments is called a "profile" and the same content may be associated to different profiles, which may differ in bit rate, codecs and resolution. These profiles are stored in a manifest file and solve the resolution problem detected in the progressive streaming.

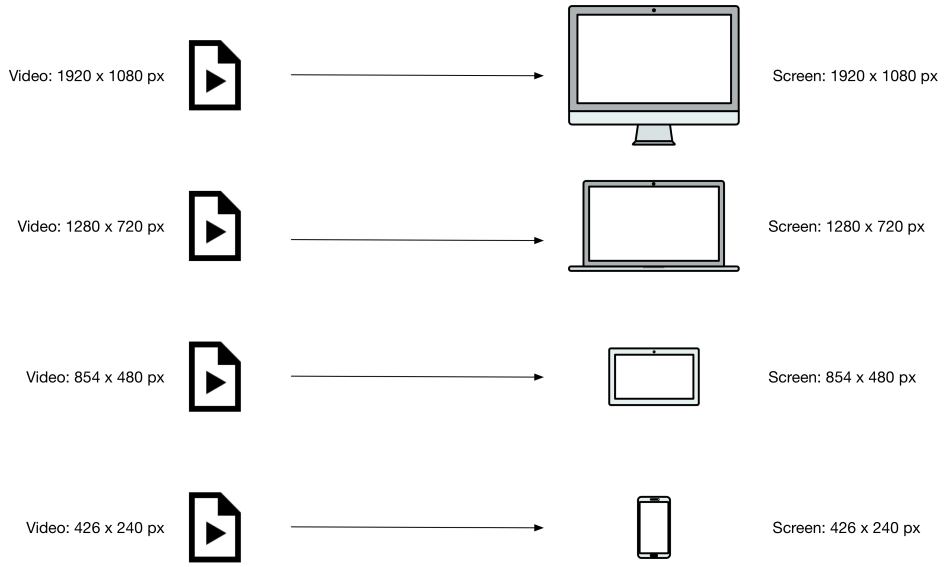

Figure 2.5: Adaptive Streaming [7]

This solution also solves the buffering problem. As previously stated, the Internet connection is the main cause for buffering and with ABR, the video stream will switch

to a smaller video files size to keep the video playing, and the video quality will vary depending on the connection to network.

For example, an user with fast Internet connection plays the video in high quality with no buffering. However, if this user changes its location and the Internet connection degrades, the device will play the video at a medium quality with no buffering [7].

2.1.4.1 DYNAMIC ADAPTIVE STREAMING OVER HTTP

Dynamic Adaptive Streaming over HTTP (DASH) is an ABR implementation designed by the MPEG group. It works over HTTP and follows a segment-based approach, dividing content into segments of short duration. The content is stored in a HTTP server in two sections, where the first one is the Media Presentation Description (MPD), which describes a manifest of the available content, their URL addresses and characteristics, and the second is the segments, which contain the multimedia bitstreams in the form of "chunks", in single or multiple files [9][10].

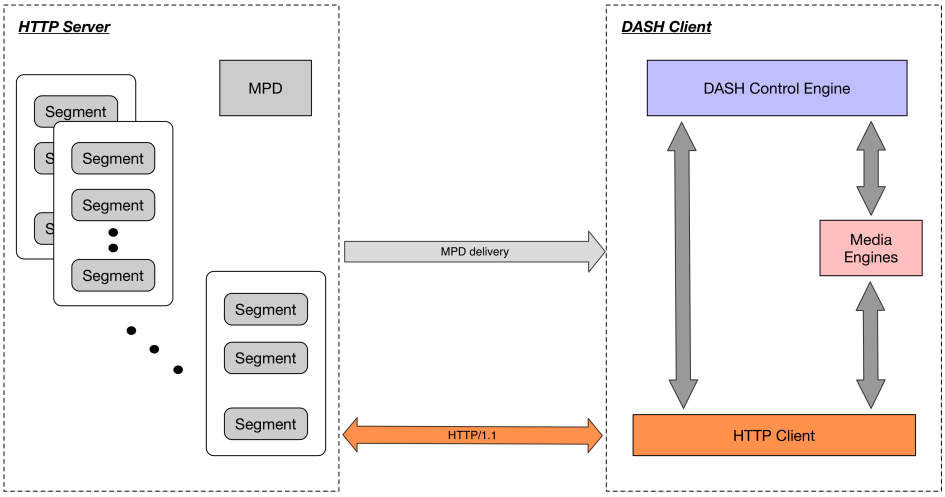

Figure 2.6: Scope of MPEG-DASH Standard [9][10]

Initially, the DASH client obtains the MPD, which can be delivered using HTTP, email, among others. The client will parse the MPD, in order to learn about the content availability, resolutions and required bandwidths. Then, using all of this content information, the DASH client chooses the appropriated alternative and starts streaming the content by getting the segments using HTTP GET requests.

During the streaming, the client monitors its network bandwidth and depending on these measurements, it decides how to adapt to the available bandwidth, using different alternatives, maintaining an adequate buffer.

DASH is available natively on Android, Smart TVs and Chromecast, and it is supported by YouTube and Netflix[7] platforms, multiple clients (VLC[8], bitmovin[9]) and services (Amazon Web Services (AWS)[10], Akamai CDN[11]).

2.1.4.2 HTTP LIVE STREAMING

HTTP Live Streaming (HLS) is also an ABR solution, implemented by Apple Inc., and lets the user send audio and video over HTTP from a web server for playback on iOS-based devices, such as iPhone, iPad and Apple TV. It resembles DASH in that it works by splitting the stream into a sequence of small HTTP-based file downloads, where each segment contains only a "chunk" of few seconds [11].

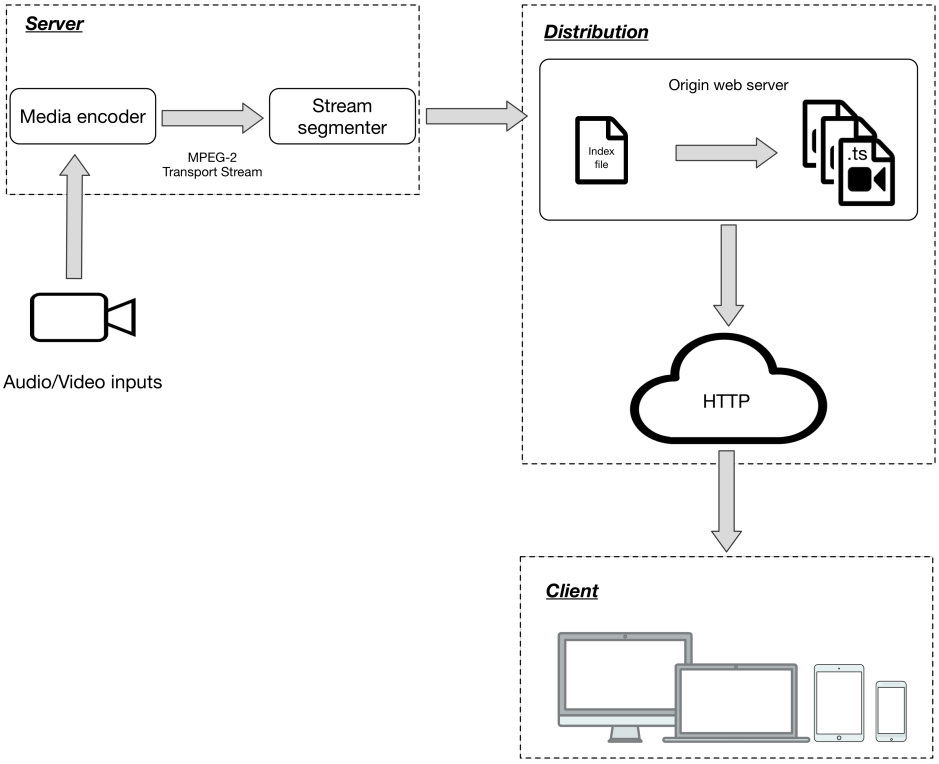

Figure 2.7: Basic configuration of a simple HLS [12]

HLS is made up of three parts: server component, distribution component and client software. The first one (server component) is responsible for taking, encoding and encapsulating input streams in a format suitable for delivery and preparing them for distribution. The distribution component consists of standard web servers, which are responsible for accepting client requests and delivering media and associated resources

[7] https://www.netflix.com/
[8] http://www.videolan.org/
[9] https://bitmovin.com/
[10] https://aws.amazon.com/
[11] https://www.akamai.com/

to it. Finally, the client software determines the adequate media to request, downloads those resources and reassembles them, in order to present the media to the user in a continuous stream [12].

Typically, the media is encoded as MPEG-4 (H.264 video and AAC audio) and packaged in an MPEG-2 Transport Stream, which is then broken into a series of short media files, by a stream segmenter that also creates and maintains an index file containing a list of the media files. These are stored on a web server, which publishes the URL of the index file, that will be read by the client software, in order to request the listed media files [12].

2.1.4.3 MICROSOFT SMOOTH STREAMING

According to [13], Microsoft Smooth Streaming (MSS) is a hybrid media delivery method which acts like live streaming, but is based on HTTP progressive download of a series of small "chunks", allowing the media to be easily and cost-effectively cached along the edge of the network, in order to be closer to clients. Similarly to the other solutions, MSS provides multiple encoded bit rates of the same media file to allow clients to seamlessly and dynamically switch between bit rates depending on network conditions. This solution results in a smoother and more consistent playback.

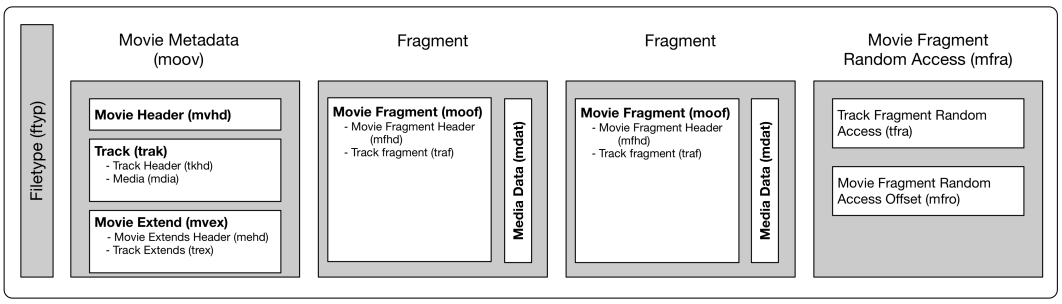

Figure 2.8: Smooth Streaming File Format [13][14]

MSS files start with file-level metadata ('moov') that describes the file. The bulk of the payload is contained in the fragment boxes that also have fragment-level metadata ('moof') and media data ('mdat'). Figure 2.8 shows two fragments, but typically one fragment exists for each two seconds of media content. Finally, the file has an index box ('mfra') that allows easy and accurate seeking inside the file [14].

2.1.5 WEB REAL-TIME COMMUNICATION

Web Real-Time Communication (WebRTC) is an open source network framework that allows real time communications in the browser [15].

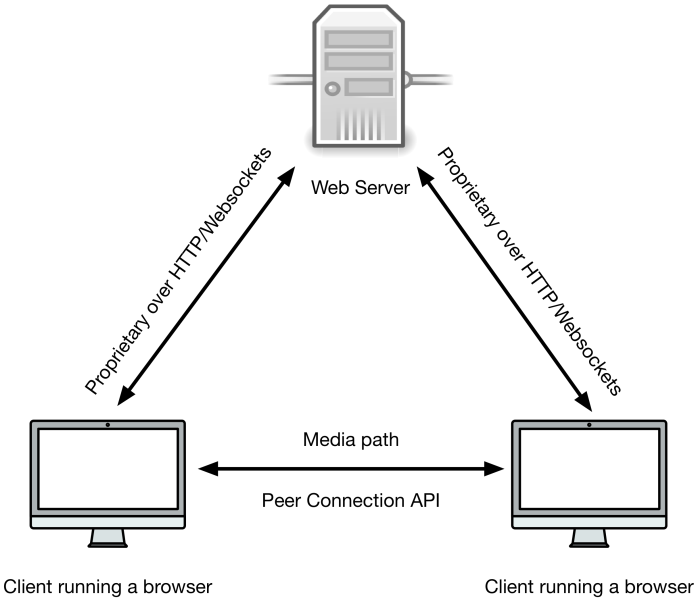

Figure 2.9: WebRTC model [16]

In the WebRTC model (Figure 2.9), both browsers are running a web application from a Web Server and a Peer Connection configures the media path, where media flows are established directly between browsers without any intervening servers. This Peer Connection allows two clients to communicate directly, and this mechanism uses the Interactive Connectivity Establishment (ICE) protocol together with a Session Traversal Utilities for NAT (STUN) server to discover the mapped IP address and port number that the Network Address Translation (NAT) has allocated for the UDP connections, and a Traversal Using Relays around NAT (TURN) server to let UDP-based media streams traverse NAT boxes and firewalls [16]. To find the best communication path, ICE allows the browsers to discover information about the topology of the network where they are deployed and its use provides a security measure, preventing untrusted web pages and applications from sending data to hosts.

This framework brings multiple benefits, such as, the users will no longer be required to download and install proprietary applications and plugins, lower bandwidth will be required and as a result, there will be less latency and high quality of audio and video. For the developers it will be easier to develop tools for communication.

The most relevant users of WebRTC are Google Hangouts[12] and Facebook Messen-

[12] `https://hangouts.google.com/`

ger[13].

2.2 SOFTWARE-DEFINED NETWORKING

In the traditional approach of networking, most of the functionalities are implemented in a dedicated appliance, such as routers, switches and application delivery controllers. Being this an hardware-centric approach, organizations are confronted with multiple limitations and according to [17], these are:

- **Traditional configuration is time-consuming and error-prone:**
  When an IT administrator needs to add or remove a single device, he will have to configure manually multiple devices and settings (switches, routers, VLANs).
- **Multi-vendor environments require a high level of expertise:**
  Typically, organizations own a variety of equipment of different vendors and to configure everything, the administrator will need an extensive knowledge of all devices.

This traditional approach is being pushed to its limit by social media, mobile devices and cloud computing. The SDN approach solves the traditional networking control approach limitations and brings numerous possible opportunities as stated in [18]:

- Support the dynamic movement, replication and allocation of virtual resources
- More easily deploy and scale network functionality
- Perform traffic engineering with an end-to-end view of the network
- Better usage of network resources
- Reduce CapEX and OpEX
- Enable applications to dynamically request services from the network
- Implement more effective security functionalities
- Reduce complexity

2.2.1 SDN ARCHITECTURE

According to [18], Open Networking Foundation (ONF) is the group that is most associated with the development and standardization of SDN, and for them, the SDN architecture is:

- **Directly programmable:** Network control is decoupled from forwarding functions

[13] `https://www.messenger.com/`

- **Agile:** Abstracting control from forwarding lets administrators dynamically adjust network-wide traffic flow to meet changing needs
- **Centrally managed:** Network intelligence is centralized in software-based SDN controllers that maintain a global view of the network, which, for applications and policy engines, is a single, logical switch
- **Programmatically configured:** SDN lets network managers configure, manage, secure and optimize network resources very quickly, via SDN programs, which not depend on proprietary software
- **Open standards-based and vendor-neutral:** SDN simplifies network design and operation because instructions are provided by SDN controllers instead of multiple vendor-specific devices and protocols.

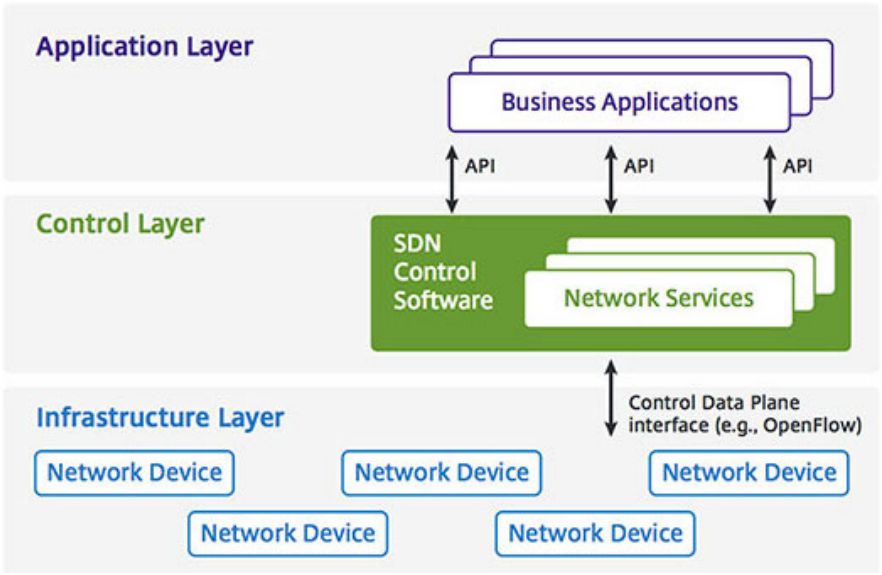

Figure 2.10: SDN architecture [18]

As shown in Figure 2.10, the SDN architecture is composed by different layers and concepts [19]:

- **Application Layer:** SDN Applications, that communicate their network requirements and desired network behavior to the SDN Controller via Northbound Interfaces.
- **Control Layer:** the SDN Controller is a logically centralized entity which translates the requirements from the Application Layer down to the Infrastructure Layer, and provides the SDN Applications with an abstract view of the network (statistics, events).
- **Infrastructure Layer:** Logical network devices, switches, that control the network forwarding and data processing capabilities.

- **Northbound APIs** are interfaces between Application Layer and Control Layer. Unlike the Southbound API (which uses OpenFlow), Northbound APIs have not been standardized yet, often recurring to REST-based protocols for their operations.
- **Southbound API** is the interface that enables communications between the Control Layer and the Infrastructure Layer and provides programmatic control of all forwarding operations, capabilities advertisement, statistics reporting and event notification. An example of a Southbound Interface is OpenFlow.

2.2.2 CONTROL PLANE & DATA PLANE

Recently, due to the lack of programmability in existing networking architectures, SDN has gained a lot of attention, because it solves this problem, enables easier and faster network innovation and separates the data plane from the control plane, which is one of the main points of interest of the SDN approach. This separation brings some advantages as:

- The software control of the network can evolve independently of the hardware
- Control behavior using programs of the Application Layer which makes debugging easier

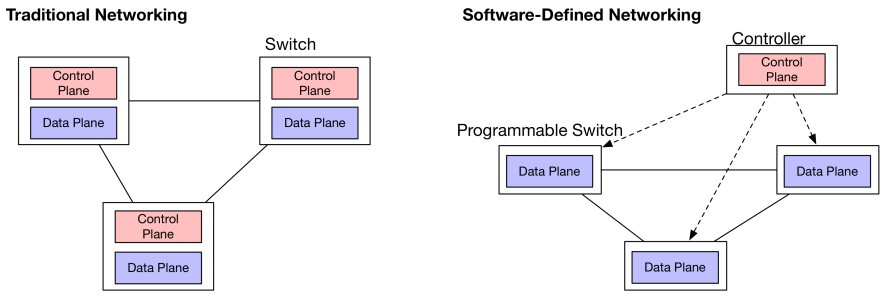

Figure 2.11: Control and Data Planes in Traditional and Software-Defined Networking

Control Plane (Control Layer in Figure 2.10) is the logic for controlling forwarding behavior, it makes decisions about to where traffic is sent. It also includes the system configuration, management, and exchange of routing table information. According to [20], it is regarded as the brain of the network. Routing Protocols and Network middle-box configuration, such as Firewall or Load Balancer configurations, are part of it.

Data Plane (Infrastructure Layer in Figure 2.10), also known as Forwarding Plane, forwards the traffic according to control plane logic to the next hop along the path to the destination, for example, IP forwarding and Layer 2 switching.

The separation of these two planes generates various opportunities enumerated in [20], such as:

- Server load balancing
- Virtual Machine Migration in Data Centers
- Adaptive traffic monitoring
- Energy-efficient networking
- Denial-of-service attack detection
- Security in Enterprise Networks

2.2.3 OPENFLOW

OpenFlow is the first standard (addressed by ONF) communications interface defined between the control and infrastructure layers of an SDN architecture (Figure 2.10). It allows direct access to and manipulation of the forwarding plane of network devices such as switches and routers [21].

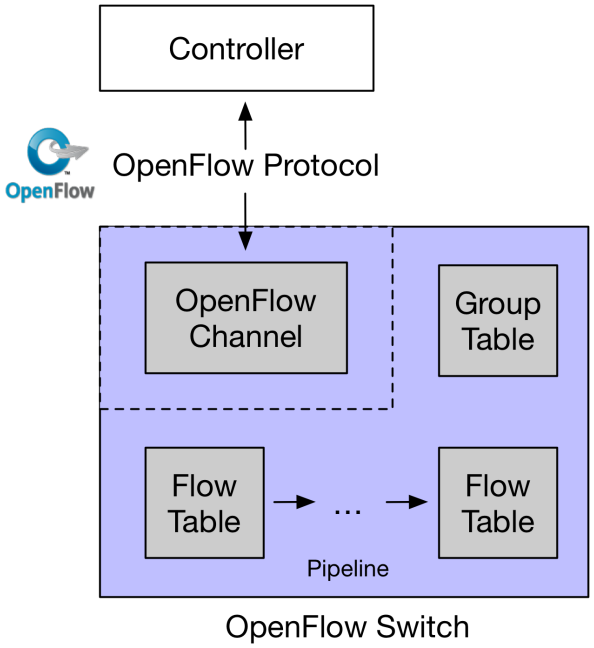

Figure 2.12: OpenFlow Switch composition [22]

An OpenFlow Switch consists of one or multiple flow tables, a group table and one or more OpenFlow channels to an external controller (Figure 2.12). The controller and the switch communicate via the OpenFlow switch protocol and by using it, the controller can do multiple operations in flow tables, as adding, updating or removing flow entries, which consist of match fields, counters and a set of instructions to apply to matching packets.

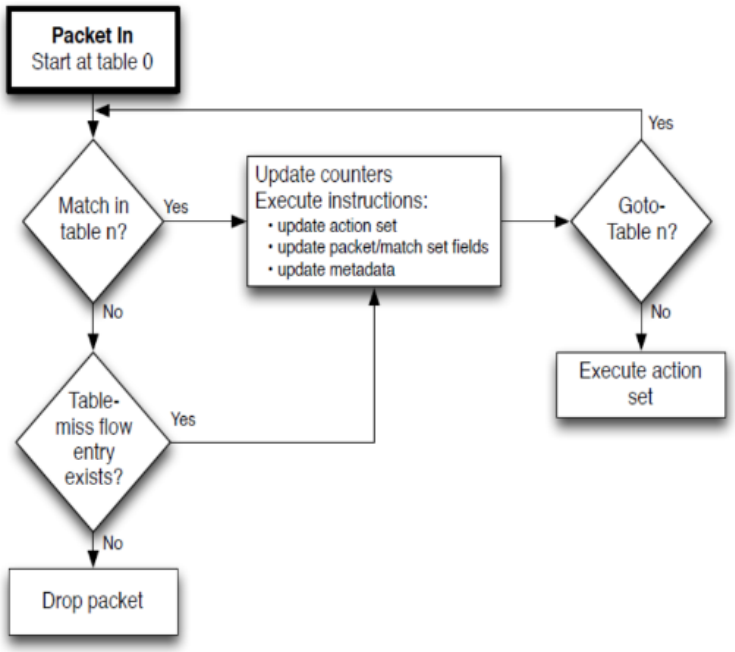

Figure 2.13: Incoming packets are matched against entries in multiple table to determine forwarding action [22]

There is a pipeline of flow tables (at least one) and the matching of a packet by a flow entry starts at the first flow table and may continue to additional flow tables. If a matching entry is found, the instructions associated with the specific flow entry are executed, else, the outcome will depend on table-miss flow entry configuration, for example, the packet can be dropped or can continue to the next flow table (Figure 2.13). The instructions associated either contain actions or modify pipeline processing. The first one describes packet forwarding, packet modification and group table processing. The second one allows packets to be sent to the next tables for further processing and allows information to be communicated between tables [22].

Finally, the group table enables OpenFlow to represent a set of ports as a single entity for forwarding packets.

2.2.4 SDN CONTROLLERS

As referred in Section 2.2.2, SDN controllers are the "brains" of the network. They act as strategic control point in the SDN network, manage flow control to switches/routers through Southbound APIs and communicate with applications to deploy an intelligent network via Northbound APIs.

Typically, an SDN Controller platform contains a collection of "pluggable" modules

that can perform different tasks. A basic example of these tasks is the inventory of which devices are within the network and the capabilities of each. More advanced tasks, such as running algorithms to perform analytics and orchestrating new rules throughout the network, are also supported.

- **POX**

  POX[14] is an open source controller developed in Python based on NOX[15] that enables rapid development and prototyping of network control software using Python scripting.

  It can also be used as a basic SDN controller by using the stock components that come bundled with it. Through the creation of new POX components, developers may create more complex SDN controllers.
- **Ryu**

  Ryu[16] is an open source component-based controller implemented in Python. It provides software components with well defined APIs that make it easy for developers to create new network management and control applications.

  Also, Ryu can be modified, extended and composed in order to create a customized controller application.

  An advantage of this SDN controller is that it supports multiple southbound protocols for managing devices.
- **Open Network Operating System (ONOS)**

  Initially, the Open Networking Lab (ON.Lab)[17] released the ONOS[18] source code, written in Java, to the open source community, but then the ONOS project joined the Linux Foundation[19] as a collaborative open source Linux project.

  ONOS provides the control plane for a SDN network, managing network components, such as switches, and running software programs or modules to provide communication services to end hosts and neighbouring networks.

  It focuses on delivering the high performance, scalability and high availability required in carrier-grade networks and service providers.
- **Open Daylight (ODL)**

  ODL[20] is an open source SDN project, written in Java, hosted by the Linux Foundation, having so far the following releases:

  – Hydrogen, February 2014

[14] `https://github.com/noxrepo/pox`
[15] `https://github.com/noxrepo/nox`
[16] `https://osrg.github.io/ryu/`
[17] `http://onlab.us`
[18] `http://onosproject.org`
[19] `https://www.linuxfoundation.org/`
[20] `https://www.opendaylight.org`

- Helium, October 2014
- Lithium, June 2015
- Beryllium, February 2016
- Boron, November 2016
- Carbon, June 2017

It is one of the few control platforms being conceived to support a broader integration of technologies in a single control platform. Therefore, according to [23], ODL goes a step beyond by providing a Service Layer Abstraction (SLA) that allows several southbound APIs and protocols to co-exist in the control platform.

Traditionally, ODL is considered to be more focused at datacenter networks.

2.2.5 OPEN-SOURCE TOOLS

2.2.5.1 OPEN VSWITCH

The OpenFlow switches implement exclusively the data plane functions, acting as a traffic forwarding device. Open vSwitch (OVS) is an open source implementation of a distributed virtual multilayer switch and it was created to enable massive network automation through programmatic extension, while still supporting standard management interfaces and protocols, such as, Netflow and sFlow.

Its use focuses on virtualized server environments and it is capable of forwarding traffic between multiple Virtual Machine (VM) in the same physical machine or between the VM and the physical network, facilitating the configuration or monitoring of the virtualized network.

As stated in [24] the bulk of the code is written in platform-independent C and is easily ported to other environments.

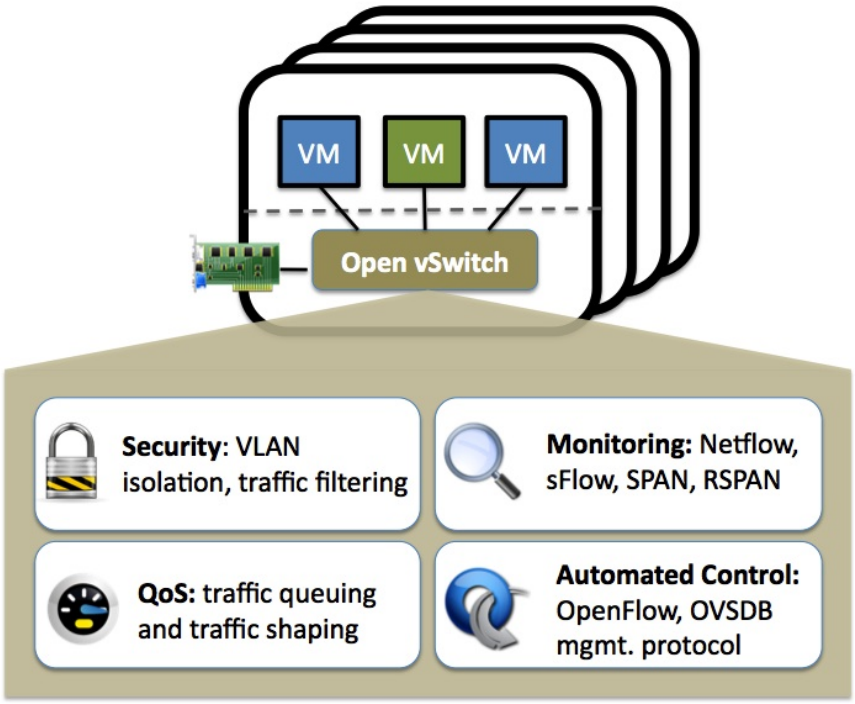

Figure 2.14: Open vSwitch overview [24]

2.2.5.2 MININET

Mininet is a network emulation orchestration system and runs a collection of end-hosts, routers, switches that support OpenFlow and links on a single machine (Linux kernel). To have a complete experimental network it supports research, development, learning, prototyping, testing and debugging and as stated in [25], brings multiple features:

- provides a simple and inexpensive network testbed for developing OpenFlow applications;
- enables multiple concurrent developers to work independently on the same topology;
- supports system-level regression tests, which are repeatable and easily packaged;
- enables complex topology testing, without need to wire up a physical network;
- includes a Command-line Interface (CLI) that is topology-aware and OpenFLow-aware, for debugging or running network-wide tests;
- supports arbitrary custom topologies, and includes a basic set of parametrized topologies;
- provides a straightforward and extensible Python API for network creation and experimentation.

Finally, it allows the experimentation of new services or applications, or modification of protocols prior to deployment in physical infrastructures and every code developed and tested on it can be moved to the real system with minimal changes.

2.2.6 SDN FOR MOBILE NETWORKS

It is expected that the mobile networks traffic grows faster than the fixed Internet traffic [26]. As previously said, the SDN paradigm has the objective of overcoming multiple limitations of the traditional network architecture and providing improvements in performance by decoupling control functions from the physical infrastructure.

According to [27], SDN has the potential to enable end-to-end unification of the control plane across wireless access and mobile core network. Then, SDN is leveraged for dynamically controlling network traffic over Wide Area Network (WAN) according to dynamic network conditions and application types. As shown in Figure 2.15, the SDN application is situated in edge nodes of wireless networks, for example, P-GW in LTE, in order to be fed by flow information, such as Client ID, Application type, Priority and QoS requirements, and enforces QoS through the multiple SDN controllers existent on network.

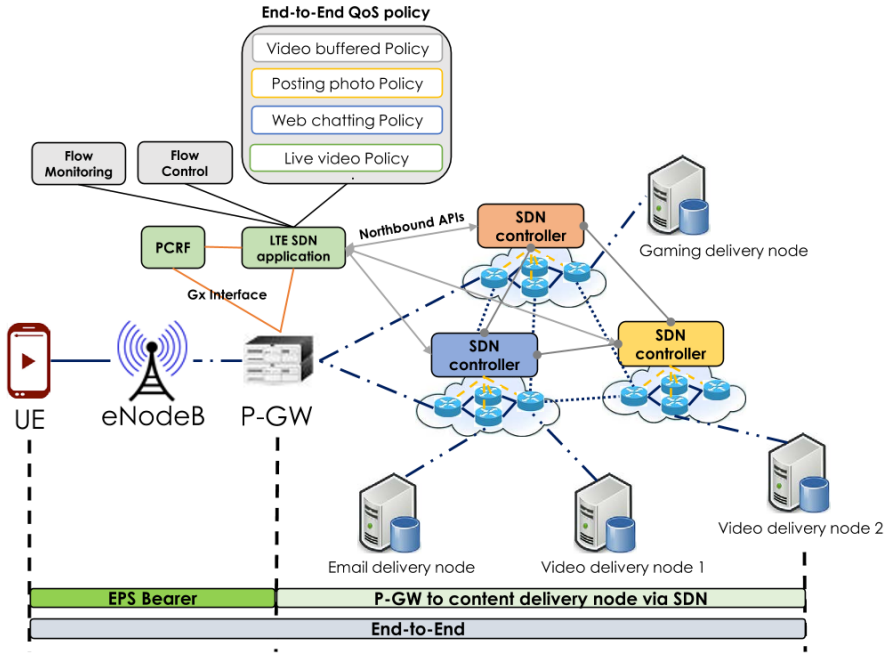

Figure 2.15: Intelligent content delivery system over LTE using SDN [27]

In [28] an adaptation of 3GPP's LTE architecture is proposed, in order to extend the Mobility Management Entity (MME) with SDN Controller functions, and P-GW, S-GW and eNodeB acts as SDN switches (Figure 2.16).

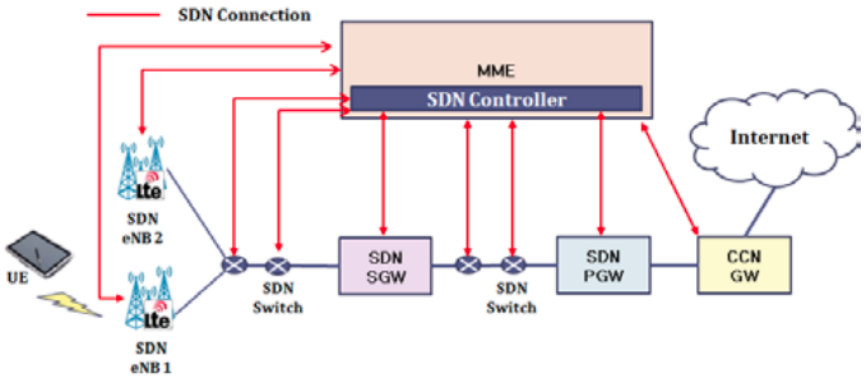

Figure 2.16: Proposed Overall Architecture [28]

2.2.7 SDN FOR VIDEO DELIVERY

In order to privilege video delivery in a SDN network, taking into account the improvement of video service, some proposals have appeared.

The authors in [29] propose a dynamic video server load balancing function for SDN controllers, in order to improve the video service QoS when network congestion occurs. There are two factors defined that affect the video server selection, which are packets loss between the server and the client and delay/delay variation depending on the server distance from the client.

In [30], QoS flow monitoring was presented as one possible SDN Application, through the extraction and monitoring of network traffic from the data plane. With the objective of leveraging the advantages of moving the sampling function from dedicated hardware into cheap standard network hardware using SDN, a prototype, named MonSamp, was implemented (Figure 2.17).

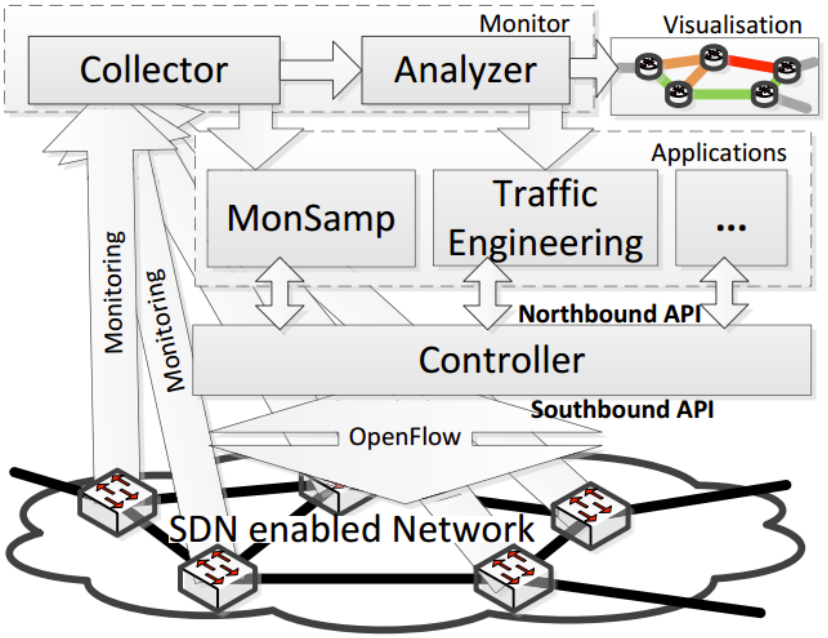

Figure 2.17: MonSamp: SDN with integrated monitoring application [30]

In a brief description, MonSamp receives information from the collector about their workload and using this information it can immediately decide to reduce the number of flows, in order to avoid random drops of packets.

2.3 QUALITY OF SERVICE

Some applications running on the network are sensitive to packet loss and/or delay. QoS refers to the capability of a network to provide better service to certain flows of traffic, with the main goal of prioritizing them without the others failing [31]. By giving more bandwidth or priority to some important flows, it is possible to manage the network, in order to distribute the bandwidth according the requirements of each flow.

To provide low delay and packets loss, it is necessary to configure some QoS components in the data infrastructure, namely [32]:

- **Classification and marking**

  Consists on the identification of packets or traffic flows and the assigning of parameters within the packet headers in order to group them. The identification can be based on IP addresses (source and destination) or network interfaces and, once a packet is identified, it can be marked via different methods, such as, IP

Precedence bits, three most significant bits within a Type of Service (TOS) byte of an IP header.

- **Congestion management**

  Aims to control congestion, through the utilization of queuing methods that prioritize traffic based on their classification and characteristics. One of the most common queuing methods is priority queuing, where the high-priority queues are served first. This method uses four different queues, High, Medium, Normal and Low.

- **Congestion avoidance**

  Prepares the network for congestion on bottlenecks throughout the data network. This implies the drop of traffic during a congestion, but it is necessary to know which packets will be dropped. There are different techniques, such as Tail Drop or Random Early Detection (RED). The first one drops every arriving packets when the queue is filled to its maximum, until it has enough space to receive new packets. The second one monitors the average queue size and drops packets based on statistical probabilities.

- **Policing and shaping**

  Limits the traffic flow. Policing drops and remarks all traffic that exceeds the limit. Shaping regulates the traffic back to a defined rate by delaying or queuing it.

- **Link efficiency**

  Fragments large data packets on slow links to reduce delay. An example of a technique that fragments a large packet into five smaller packets and allows a voice packet to be sent between them, in order to reduce the delay, is Link Fragmentation and Interleaving (LFI).

2.3.1 QOS IN DIFFERENT ACCESS TECHNOLOGIES

2.3.1.1 LTE ACCESS

For enhancing service quality according to the user profile and application service type, LTE provides a comprehensive QoS, policy control framework and charging system.

Considering Figure 2.18, QoS is implemented between the User Equipment (UE) and Packet Data Network Gateway (P-GW) and is applied to an Evolved Packet System (EPS) bearer, which is a set of network configurations that provide special treatment to a set of packets. It is composed by the Radio bearer, S1 bearer and S5/S8 bearer.

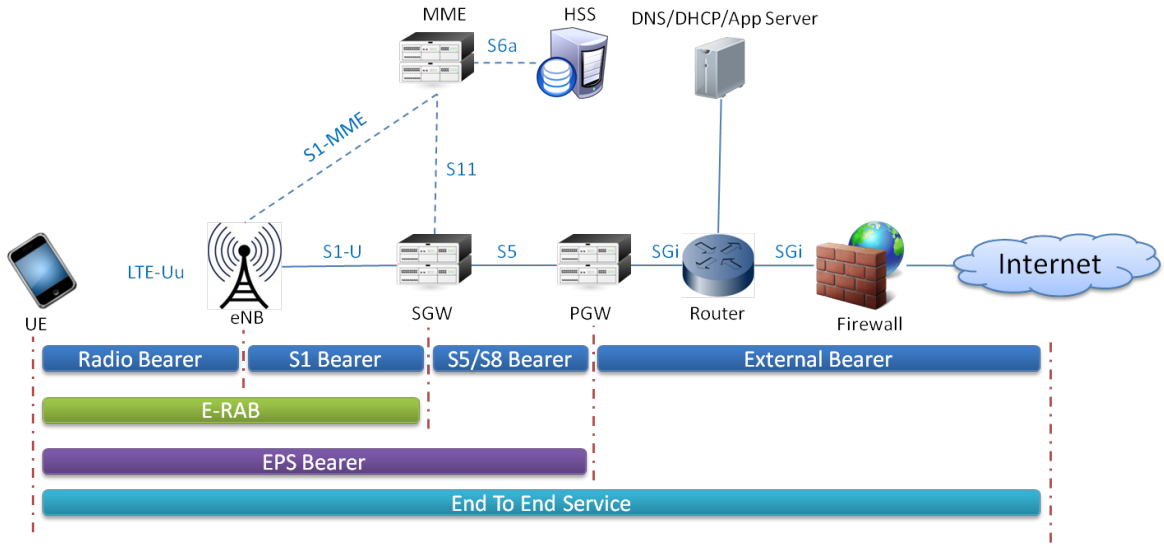

Figure 2.18: LTE Network Architecture[33]

There are two types of Bearer, Dedicated and Default and at least one default bearer is established when the UE attaches to the LTE network. When QoS provisioning is required for a specific service, such as video, the dedicated bearer is established.

Figure 2.19 shows the QoS parameters characterizing Bearers in LTE. QoS Class Identifier (QCI), is a parameter for all bearers, which consists in the differentiation of bearer traffic to have different QoS. Guaranteed Bit Rate (GBR) bearers have GBR and Maximum Bit Rate (MBR) as parameters, with the first being, the minimum guaranteed bitrate, and the other, the maximum as the name implies. Finally, the non-GBR bearers have Access Point Name Aggregate Maximum Bitrate (APN-AMBR), which is the maximum allowed total non-GBR throughput to specific Access Point Name (APN), and User Equipment Aggregate Maximum Bitrate (UE-AMBR), the maximum allowed total non-GBR throughput among all APN to a specific UE.

As shown in Figure 2.19, default bearers can only be non-GBR type.

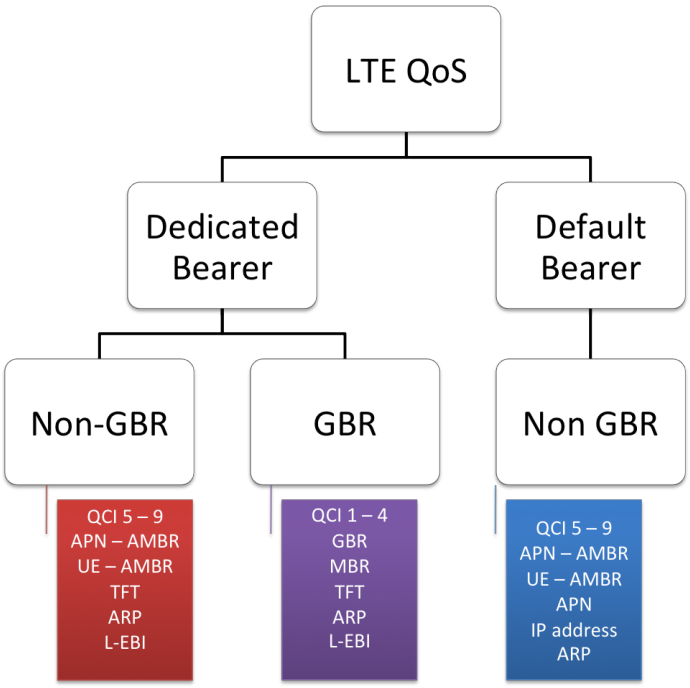

Figure 2.19: LTE QoS parameters and bearers types[33]

2.3.1.2 WLAN ACCESS (IEEE 802.11E)

IEEE 802.11e composes the QoS strategy for IEEE 802.11 networks and defines a set of QoS enhancements for Wireless Local Area Network (WLAN) applications. Distributed Coordination Function (DCF) and Point Coordination Function (PCF) are the two modes used by the original IEEE 802.11 standard. In the first mode, which does not allow traffic differentiation, multiple stations compete for the medium and rely on Carrier Sense Multiple Access – Collision Avoidance (CSMA/CA) and Request to Send (RTS)/Clear to Send (CTS) mechanisms. In the second one, mobile stations are coordinated by an AP, similarly to the role of eNodeB in LTE networks. This mode is splitted in three phases: Beacon transmission, Contention Free Period (CFP) and Contention Period (CP). Initially, in Beacon stage, the AP is responsible for sending beacon frames. In CFP, the AP informs each user about the right to transmit a packet, while in CP DCF is used.

Both modes of the original 802.11 standard are enhanced by 802.11e, introducing a new coordination function, Hybrid Coordination Function (HCF) and adding Traffic Categories (TC), which assign different priorities to the traffic. The two proposed modes, based on DCF and PCF, are Enhanced Distributed Channel Access (EDCA) and HCF Controller Channel Access (HCCA) respectively.

2.3.2 QOS IN VIDEO STREAMING

Video streaming applications require QoS, in order to give users the highest video quality. During a video stream, and according to [34], there are some minimal aspects that should be achieved, such as, the loss percentage should be less than 5%, and latency/delay, depending on the application buffering capabilities, should be as low as possible.

The main requirement for the video streaming applications is the quality and not only the delay. It is necessary to ensure that, in case of network congestion, the video packets have priority and are the first to be sent, in order to reduce the loss percentage and give receivers the best quality possible.

With the QoS requirements for video streaming, it becomes necessary to classify different types of services separating them, such as voice, video and special data applications, and the marking of traffic should be as close to the source device as possible.

2.3.3 QOS IN SOFTWARE DEFINED NETWORKS

As stated in [35], QoS in traditional network architectures is a constant problem. SDN appeared to respond to some limitations of traditional architectures. Authors also reference some aspects where SDN can help regarding to QoS:

- QoS-motivated routing is a function that SDN can help networks to improve, because it allows per-flow routing and network operators to use various routing algorithms, instead of the typical shortest path, generating forwarding tables that manage different isolated flows. Also, the dynamic routing of flows is possible due to the decoupling of control and forwarding functions. With these abilities, more QoS-motivated routing mechanisms will appear;
- powerful and easy-to-use automated QoS management frameworks can be created by the network operators through resource reservation, queue management and packet scheduling. Compared with the traditional network architectures, QoS configuration is easier in SDN;
- network monitoring helps detecting problems in real time and can predict future events in a network. SDN offers a network manager the possibility of monitoring a set of metrics per packet, per port, per table, among others;
- through SDN, QoS can be provided in multiple ways, such as via QoS policy management and content delivery mechanisms based on its characteristics, for example, per-flow control concept.

To summarize, the SDN paradigm brings many functions that improve the use of QoS, which help network managers in monitoring, traffic differentiation and network management.

2.4 WIRESHARK - NETWORK PROTOCOL ANALYSER

Primarily, named Ethereal, Wireshark is an open source tool for packet analysis that uses libpcap, a low-level packet capture library developed by the tcpdump[21] developers. It allows that the user can see what is happening on its network at a low level. In addition to have a graphical front-end, which improves the user interaction, it includes a large quantity of features, stated in [36], such as:

- deep inspection of multiple protocols, with constant addition of new protocols support;
- live capture and offline analysis;
- runs in multiple platforms, such as Windows, macOS, among others;
- use of display filters;
- live captures can be done in any interface (Ethernet, IEEE 802.11, Bluetooth), depending on the user's platform;
- decryption support for some protocols, like SSL/TLS, IPsec, among others.

[21] http://www.tcpdump.org/

CHAPTER 3

Solution

The purpose of this chapter is to present a solution for a real-time network monitoring application that detects video flows through the capture of network packets (Monitor App), in order to attribute QoS policies and priority to these flows. The implementation of this application will be presented in the following chapter.

As this application is part of a PoC project named VDSNet, Section 3.1 will provide details regarding its description, architecture and the role of Monitor App.

In Section 3.2, a description of the Monitor App and its objectives are introduced. Then, in order to ensure reliability and scalability, there are some requirements (Subsection 3.2.1) that the application must have to satisfy its stakeholders, addressed in Subsection 3.2.3. Also, different use cases will be identified in Subsection 3.2.4. Finally, in the last Subsection 3.2.5, the application's architecture is shown with a description of each component that composes it.

3.1 VDSNET PROJECT

The VDSNet Project focuses on video streaming scenarios where mobile users are the video senders, and monitoring applications and SDN technologies are positioned for improving these services. VDSNet considers the importance of having low latency during the stream, and as such, it focuses on the improvement of the uplink operation, through the application of QoS policies and priorities to the network, ultimately improving the user experience.

As a main goal, the project focuses on the detection of video flows in the network using a monitoring application, and their prioritization through a SDN environment.

A real world application of VDSNet is during a concert with thousands of people where many of them intend to do a live stream simultaneously, where the traditional network does not have conditions to support this quantity of mobile network traffic. VDSNet aims to detect, in real-time, which flows are correspondent to video and, through SDN, give priority to them, in order to provide a good quality of video to the receivers, instead of a low-definition image.

3.1.1 ARCHITECTURE

Knowing the main objectives of the VDSNet project, it is now necessary to understand its architecture and components in order to be aware of what each one does. The architecture diagram is shown in Figure 3.1 and is composed by the multiple components described below.

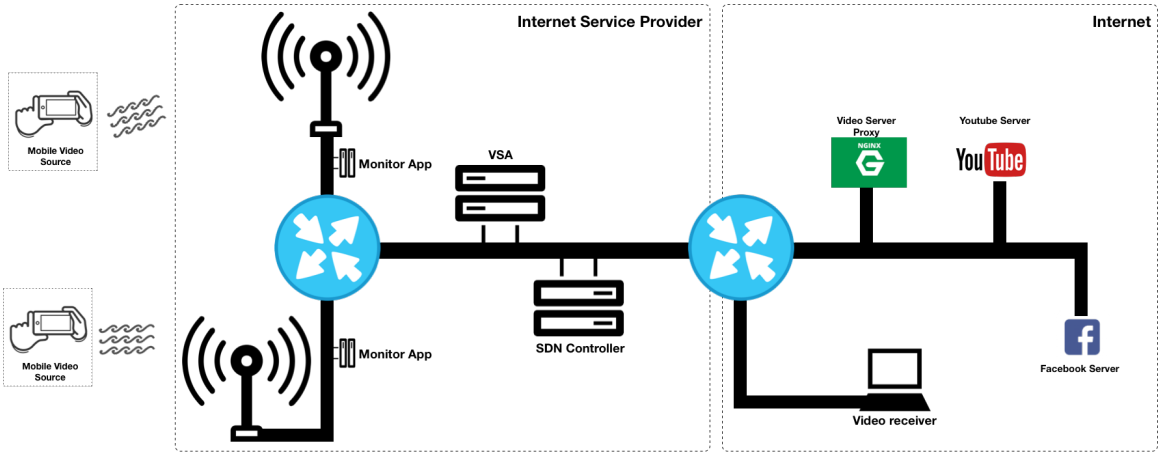

Figure 3.1: VDSNet - Architecture Diagram

- **Mobile Video Source**

  It is the mobile application used to do the live stream by the sender. It can be any application capable for the effect, such as, Facebook, YouTube or a simple RTMP live stream application.

- **VSA**

  This component is responsible for dynamically transmitting video service requirements to the SDN Controller. It contains an Application Programming Interface (API) that can be used by applications like the Monitor App to send requests related with the service.

- **Monitor App**

  Monitor App can be situated in a server at the entrance of an ISP, in a BS or in an AP, and is responsible for the detection of video flows and for notifying the VSA when one is detected or finishes.
- **SDN Controller**

  As previously stated in 2.2.4, the controller is the "brain" of the network. It manages switches/routers flow control through the Southbound APIs and communicates with applications, such as the VSA, to deploy an intelligent network via Northbound APIs.
- **Video Receiver**

  Video receiver is the application connected to the video server. It can be, for example, the Facebook website, the YouTube application or the VLC[1] application transmitting a stream via an RTMP url.
- **Video Server Proxy, Youtube Server, Facebook Server, among others**

  These are the servers to where the video is sent to by the Mobile Video Source and they are responsible for sending the video to the receivers.

3.1.2 MONITOR APP ROLE IN THE VDSNET PROJECT

This dissertation focuses on the development of an application, whose main goal is the detection of video flows through the capture of network packets. In the VDSNet project, this application has the name of Monitor App and it can be situated, as previously said, in a server at the ISP's entrance, in a BS or in an AP.

In the VDSNet concept, the application has access to all network flows that enter the ISP and only interacts with the VSA component, in order to request QoS and priority to the video flows that it detects.

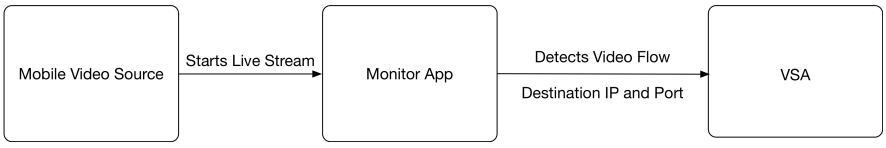

Figure 3.2: Interaction Diagram between Monitor App and VSA

As Figure 3.2 shows, an user starts a live stream and then the Monitor App detects it and notifies the VSA, sending it the destination IP and port of the detected flow.

[1] `http://www.videolan.org/vlc/index.html`

3.2 MONITOR APP

3.2.1 REQUIREMENTS

As stated, the solution is part of a PoC, VDSNet project, which is described in Section 3.1. In this section, the following requirements should agree with the objectives proposed in Subsection 3.2.2.

R1) Video Detection

Taking into account the purpose of the Monitor App, the main requirement that it should satisfy is the detection of video. The application should be able to detect video through the capture of network packets, using multiple detection mechanisms, in order to notify users about the existence of video or not.

R2) Fast Responsiveness

The main requirement to be satisfied in a real-time application or platform is fast responsiveness. If the system that integrates the Monitor App needs the information sent by it, in order to do something depending on that information, the response should be fast. However, if the system only wants the information for statistics, the response can be slower, in spite of the application preferring the fastest possible response. For example, in VDSNet, the objective is to always grant a fast response, in order to satisfy every users of the application and to optimize the network as soon as possible.

R3) Efficiency

Being the Monitor App an application of detection, in this case video, it should be efficient, in order to have a great percentage of certainty when it detects something. It is important that the application fails as little the detection of flows as possible. In a case as VDSNet, if the detection misses a video flow (i.e., false negative), a flow that would have priority for transmitting video will be considered as a normal flow, which is critical taking into account the purpose of the project.

R4) Performance

Performance is an important requirement in this solution, due to the possibility of running the application in multiple types of devices. The Monitor App should be able to work with a large number of flows without losing performance, when it is running in a server or in a BS with limited resources. Therefore, if the machine does not have

resources to work without losing performance, the application is easily scalable, as described below.

R5) Flexibility and Scalability

Other important requirements are the flexibility and scalability of the application. It should support easy integration with other applications, because it is a network application to be integrated on different types of systems, such as VDSNet, interacting with VSA, or monitoring applications, sending more detailed information. Moreover, as a consequence of the growth of number of flows, scalability should also be supported. So, as explained in Subsection 3.2.2, adding more nodes to the system, with one of them becoming the responsible for receive all the flows and requesting the analysis of some flows to other nodes, is a requirement.

R6) Failure Handling

Another requirement is failure handling. The application should be able to recover from faults, and from the last state since then. For example, if the application fails and loses connectivity to a database when restarting, it should have access to the last state, in order to use the information that it had in cache when the fail occurred.

R7) Security

Finally, the last requirement is security. It is important that all connections to the databases are secure in order to avoid any interception of data during a connection. The system should be prepared and configured to only accept secure connections, using certificates.

3.2.2 OVERVIEW AND OBJECTIVES

The Monitor App's purpose is a network monitoring application focused on detecting video flows in the network, in order to notify the applications that are using it, about the existence of a live stream. It is composed by five different detection mechanisms, namely:

- **Whitelist**: the whitelist contains flows previously defined as video, allowing the system to respond immediately notifying the existence of video.
- **Blacklist**: the blacklist contains flows previously defined as non-video, allowing the system to respond immediately notifying the non-existence of video.

- **Network Port Analysis**: analysis of the network port number of a flow that some protocols have as fixed, for example, RTMP always uses the TCP port number 1935.
- **Hostname Analysis**: the hostname is analyzed through the destination IP, in order to detect the existence of some keywords, for example, "stream" or "livestream".
- **Periodicity Analysis**: the frequency of the incoming packets is analyzed in order to obtain the periodicity graphic and then compare it with previously stored samples.

It can be configured to be used as monitoring application by multiple types of applications, for example, a monitoring platform that shows the percentage of video flows in a specific network, or a real-time platform that notifies the user when a video flow is detected. Considering these examples, the application can be used by ISPs, which are the main target of the VDSNet, but it can also run on an enterprise network, where it is possible to control the use of video inside the enterprise, and in a domestic network, if the owner wants to monitor its network traffic. Thus, it can have a professional usage, like ISPs and enterprises or a personal usage, using it at home.

Another advantage of this application is its possibility to be placed in many different spots, such as servers, BSs and APs, and the only requirement is having access to all network packets. For example, an ISP does not need to waste CPU resources in a BS if it can install and run the application on a server, with better processing characteristics, at its entrance. So, the user places the Monitor App where it wants, in order to do an optimal management and usage of its resources.

Being a scalable solution, it also allows the distribution of the work by multiple machines, instead of doing everything in a single one. When a flow enters the main machine, it can ask another one with access to network packets in real time to do the analysis of that flow. Then, the user can have a main machine to receive all the flows, that only forwards the work to other ones, instead of doing it (Figure 3.3). Thus, the Monitor App can run using a single machine or multiple machines, where the primary is the one that receives all the flows, and the secondary machines do the analysis of a certain flow.

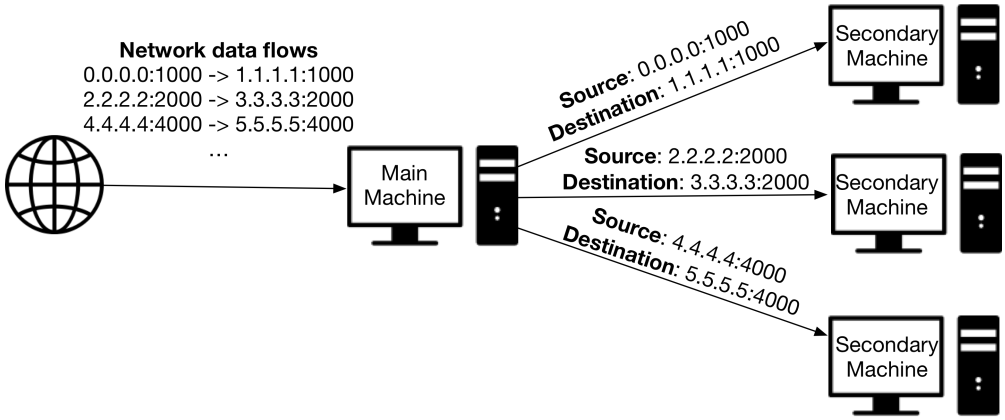

Figure 3.3: Flow distribution example

In a real-time system, latency is a critical point, so this application implies a small latency time since the live stream starts with only the periodicity analysis taking more time because it is a mechanism that needs a sample of a few seconds. The scalability and the work distribution by different machines fit in this issue, so is expected that it will free the main machine CPU, which probably will reduce the latency.

To summarize, this application is not limited to its usage in SDN paradigms to control the flows of a network, as in VDSNet. It can also be deployed in traditional networks, and can be used to feed with data monitoring platforms too.

3.2.3 STAKEHOLDERS

A dependency for the success of any project is the people involved on it, either by being integral part of the project's execution, or by being the final users. In this subsection, the stakeholders will be identified, in order to understand their expectations.

As the Monitor App can be used in multiple different scenarios, the stakeholders vary with the context where the application is.

- Developers team
- ISP network management team
- Enterprise management team
- Techsavvy[2] users
- Live streamers
- Monitoring platforms developers

[2]Person that knows a lot about modern technology, especially computers.

Depending on the scenario where the Monitor App will be used, the primary stakeholders vary. In the VDSNet project, the ISP network management team and the live streamers are the main ones. First by being the ones who will take advantage of the application features, either for a SDN paradigm where they want to control the flows or for a traditional network architecture. Second, by taking benefits when they do a live stream getting better quality and priority for their streams. However, if the utilization scenario will be an enterprise network or a home network, the main stakeholders will be the Enterprise management team or the techsavvy users, depending on the scenario. The monitoring platform developers can also be interested in this application, because it gives data to feed their platforms, making them more complete.

Finally, the developer team has interest in creating a system that satisfies all the stakeholders, and upgrade it, in order to make the project a success.

3.2.4 USE CASES

In order to understand the features offered by the Monitor App, and its interactions with the components and users of different scenarios where the application can interact with, different use cases are presented in this section. Firstly, the use cases related with the VDSNet project will be shown and then other possibilities, depending on the utilization of the application. The first use case, described below, is equal for all uses of the Monitor App.

a) Live Streamer starts a stream (VDSNet and general application)

The first situation is when a user starts the interaction with the system where the Monitor App acts. In this case, we can consider that moment when a new live stream starts. As shown in Figure 3.4, after the beginning of the live stream, the Monitor App will verify the existence of video, using one of the five detection mechanisms presented in Section 3.2.2.

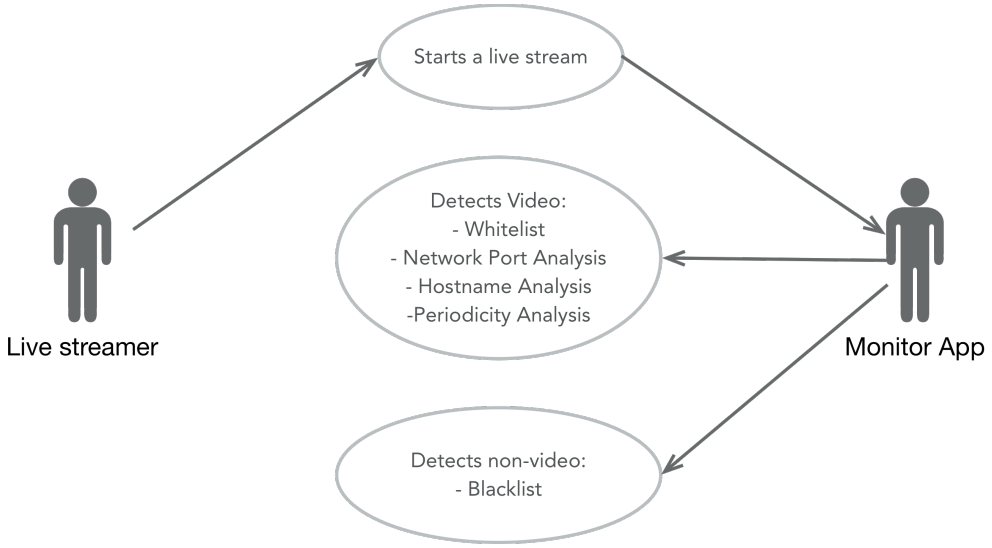

Figure 3.4: Use case diagram: Live Streamer starts a stream

b) Monitor App informs VSA if video was detected or not (VDSNet)

In VDSNet, the second use case occurs when the Monitor App analyzes a flow of network packets (Figure 3.5). There are two possible actions that the Monitor App can do. The first one addresses cases where video flows are detected, and the application informs the VSA about the existence of these flows. The second occurs when a flow is not video and the Monitor App informs VSA, in order to remove some QoS policies previously attributed to that flow.

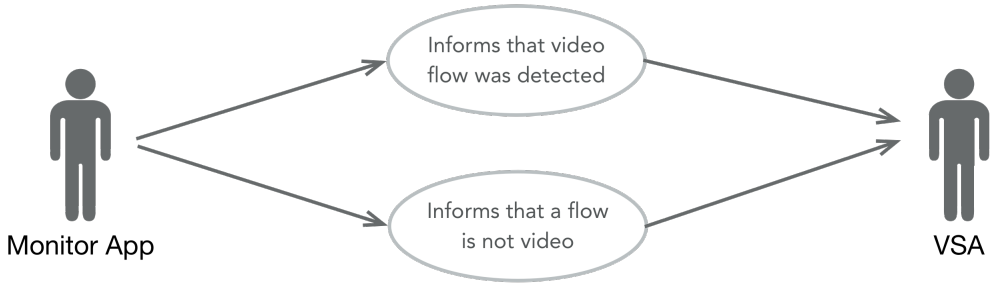

Figure 3.5: Use case diagram: Monitor App informs VSA if video was detected or not

c) Monitor App sends data to monitoring platforms

Another use case occurs when an external monitoring platform wants data to present to its users (Figure 3.6). Basically, the Monitor App sends the information of an analyzed flow, whether it is video or not, providing the percentage of certainty when the used detection mechanism is the periodicity analysis. With these types of information,

the monitoring platforms can use the data in multiple ways, for example, the percentage of video flows in a network.

Figure 3.6: Use case diagram: Monitor App sends data to monitoring platforms

d) Monitor App notifies network manager

In the same way as the use case shown in Figure 3.5, this use case (Figure 3.7) corresponds to the moment that the Monitor App notifies a network manager of a system using the application about the existence of video.

Figure 3.7: Use case diagram: Monitor App notifies network manager

e) Main node (Monitor App) requests a secondary node to analyze a certain flow

The last use case derives from the scalability importance stated in Section 3.2.2. When the deployment uses multiple machines to distribute the work, this use case occurs. So, the main node, that receives all flows, requests a secondary node to do the analysis of a certain flow (Figure 3.8). This use case will free a lot the main node, avoiding the realization of any slower analysis, such as, Hostname Analysis or Periodicity Analysis (Section 5.3).

Figure 3.8: Use case diagram: Main node (Monitor App) requests a secondary node to analyze a certain flow

3.2.5 ARCHITECTURE

After presentation of the application requirements and objectives, this subsection aims to present two possible architectures, namely centralized or distributed, by specifying the different components that constitute them and describing their interactions. A flow diagram of how the Monitor App works will also be shown in order to understand the different mechanisms that form the final solution.

Figure 3.9 presents one of the possible architectures. This architecture is composed by different components and the Monitor App runs only in one machine.

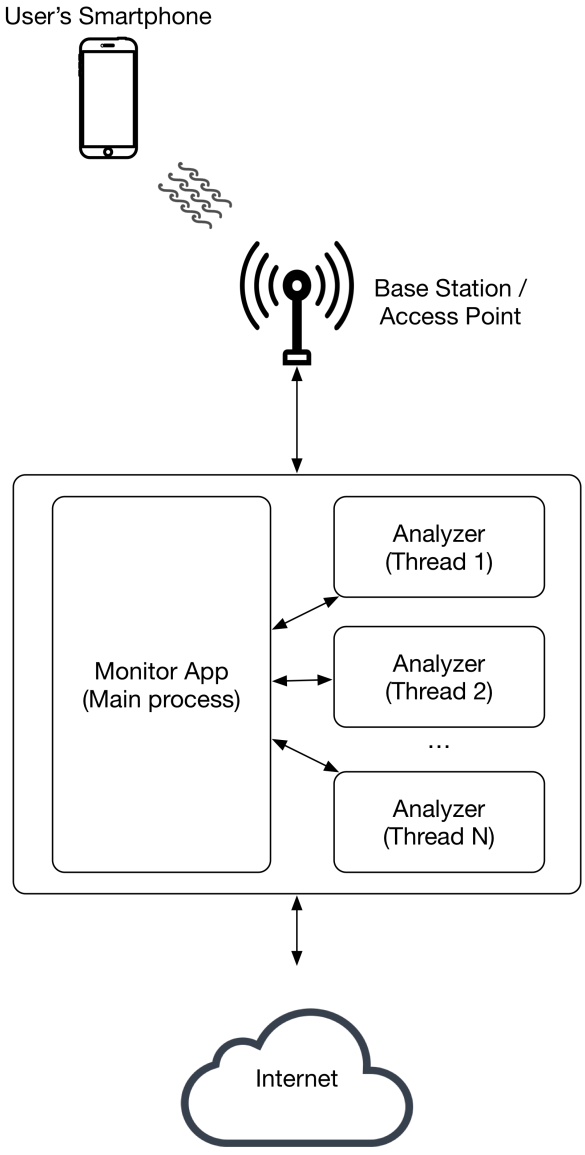

Figure 3.9: Monitor App: architecture diagram (centralized)

Firstly, the user's smartphone is the component responsible for the initiation of a live stream that will generate data flows for the Monitor App to analyze. Using an application designed for the purpose, it can transmit what it wants to whom it wants. This part of the architecture corresponds to the live streamer in the Stakeholders section (3.2.3). However, to do this, it needs to be connected to a BS or an AP, in order to have access to Internet, where the platforms servers, such as Facebook or YouTube, are hosted.

The main component of both solution architectures is the Monitor App. It receives the data flows and analyzes them in order to understand if a certain flow corresponds to a video stream or not. This component is divided in two modules, the main process and the analyzers. The first one is responsible for receiving all the flows and verifying if the destination of a flow is in a whitelist or blacklist, or if the destination port is defined for some streaming protocol, such as RTMP. If the flow is not in the whitelist, blacklist or the port is not known, the main process launches a thread to conclude the analysis, either the Hostname Analysis or the Periodicity Analysis, if needed. These threads will process the flows, in order to free the main process to start the processing of new flows.

Contrarily to Figure 3.9, Figure 3.10 presents an architecture where the Monitor App modules run in multiple machines (distributed). The main node is equivalent to the main process of the architecture presented in Figure 3.9 and the remaining nodes do the same as the threads. This solution allows for a scalable approach and frees resources from the main node, improving overall performance.

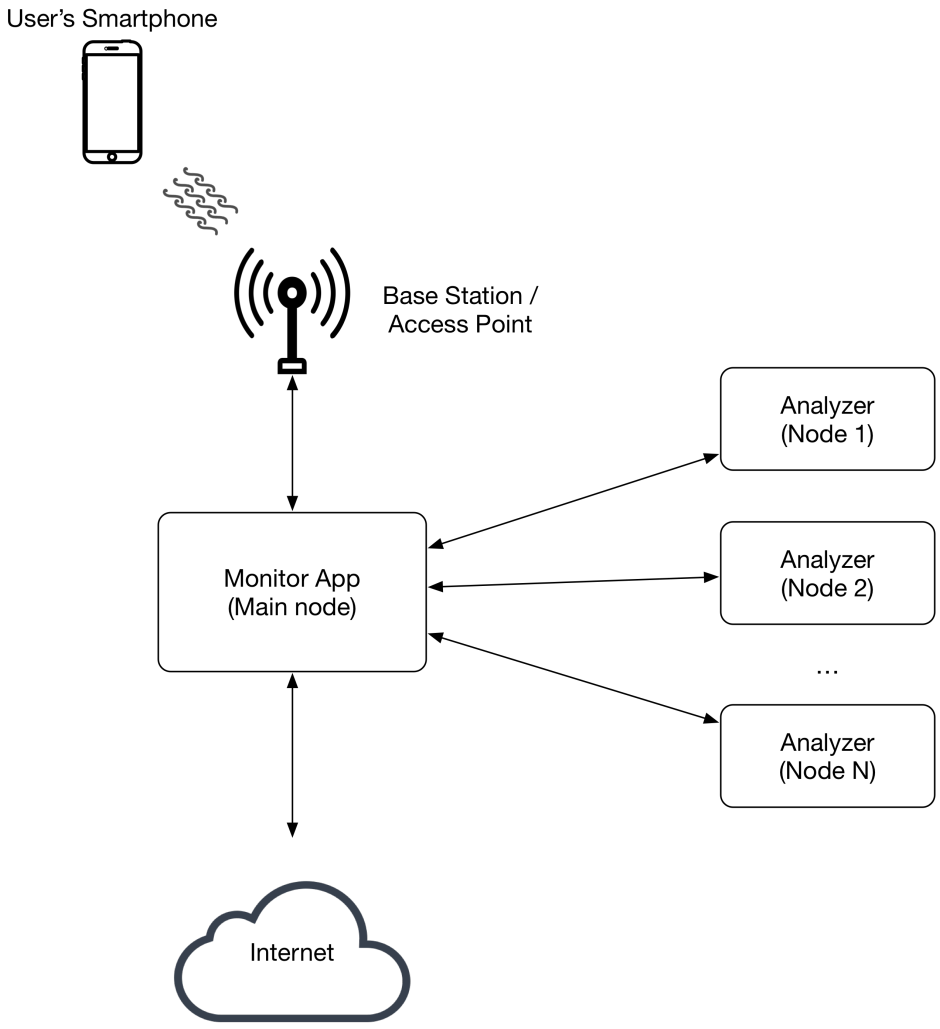

Figure 3.10: Monitor App: architecture diagram (distributed)

Finally, the operational process inside the Monitor App is shown in Figure 3.11 through a flow diagram. Initially, a flow enters the Monitor App, which will verify if that flow is in a whitelist containing the known video flows, in a blacklist with non-video flows, or in neither. Then, if it is not in the whitelist or in the blacklist, the application will analyze the destination port, in order to understand if that port corresponds to a video streaming protocol. As the whitelist and the known ports are in cache, these detection mechanisms are executed in the main module, because it is an immediate process. After these processes, the application sends information about the flow to an analyzer, that could be a thread or a node, depending on the architecture. The analyzer starts the Hostname Analysis, which corresponds to a search of keywords in the hostname. An example of a keyword that the Monitor App considers video is "livestream", but the application also analyzes keywords that do not correspond to video, such as "dropbox". The last mechanism is the Periodicity Analysis, which implies a

capture of a few seconds of the flow to determine the periodogram corresponding to the frequency of packets from a certain flow. This analysis provides, as response, the probability of the flow being video.

In brief, the solution presented in this chapter could adopt one of two different system architectures that will not change the flow of Monitor App.

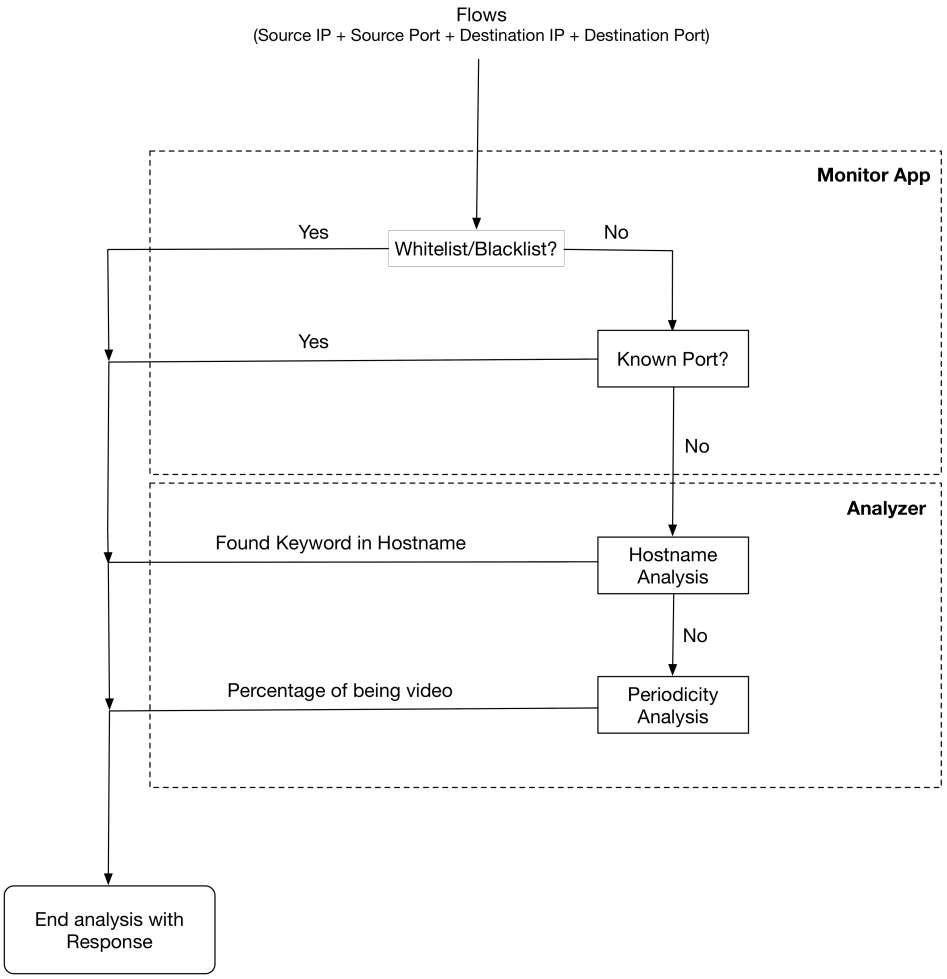

Figure 3.11: Monitor App: flow diagram

CHAPTER 4

Monitor App Implementation

In order to provide a video monitoring network application, Chapter 3 presented a possible solution, contextualizing it in the VDSNet project and addressing all requirements, stakeholders, use cases and architectures.

This chapter aims to describe the implementation of the Monitor App, explaining the steps taken and the reasoning behind each decision. In Section 4.1, the adopted technologies are exposed and their selection is explained. Then, in Sections 4.2 and 4.3, the architecture of the Monitor App implementation, and the different mechanisms developed to form it, are described. Initially, a description of how Whitelist and Blacklist mechanisms work is presented in Subsection 4.3.1. After that, in Subsections 4.3.2 and 4.3.3, the Network Port Analysis mechanism and the Hostname Analysis are explained. Finally, an explanation of how the Periodicity Analysis is done is addressed in Subsection 4.3.4.

In this work, only one of the two architectures purposed in Chapter 3 was implemented, corresponding to the one presented in Figure 3.9, where every module is running in a single machine (centralized). However, the implementation and separation of different modules of the application were made to support a quick and easy transition to the architecture presented in Figure 3.10 (distributed). As shown in the flow diagram of the application (Figure 3.11), it is composed by two different modules, the Monitor App main process and the Monitor App secondary process, named analyzer. The main process is responsible for the Whitelist/Blacklist and Network Port Analysis mechanisms. The other one, analyzes the hostname and the periodicity.

4.1 ADOPTED TECHNOLOGIES

This section explains the reasons that led to the choice of the adopted technologies. The main criterion of choice was allowing the entire system to have the best performance as possible. It is composed by three subsections, with the first, describing all technologies used in Monitor App development. The second one is related with the technology adopted to deploy a distributed database to store multiple information needed by the system. Finally, the latter subsection describes the technology selected to use as Load Balancer of all databases nodes.

4.1.1 MONITOR APP

In order to provide an application with low response latency and high performance, all the modules of Monitor App were developed in C language. This choice is related with the fact that the Monitor App needs to process a lot of network flows and analyze them, so it requires the best performance possible and the lowest latency, in order to respond as soon as possible with the analyzed info about a certain flow.

Libpcap, a C/C++ library to capture the network packets, was used. This choice was based on Wireshark (Section 2.4), which uses it and has a great performance regarding to the network protocol analysis.

4.1.2 DISTRIBUTED DATABASE

Different kinds of information, such as Whitelist, Blacklist, known Network Ports, should be stored in a distributed database, which all Monitor Apps working have access to. This means that every Monitor App in a network has access to the same info and when a database update is necessary, it does not need to update all Monitor Apps and databases. In fact, in a distributed database system is only necessary to access one node of the database to update all the others, which is a great advantage.

As database system, it was chosen CockroachDB[1], a distributed SQL database. Ease to scale horizontally and simplified configuration were the great reasons of this choice. Also, it provides encrypted inter-node and client-node communications over SSL/TLS.

An important feature provided by Cockroach DB is a platform where the database manager can monitor all nodes.

[1] `https://www.cockroachlabs.com/`

This selection was result of an analysis of different Database solutions. The comparison of these solutions is presented in Table 4.1.

	Cockroach DB	**MySQL**	**PostgreSQL**	**MongoDB**
Automated Scaling	Yes	No	No	Yes
Automated Failover	Yes	Optional	Optional	Yes
Distributed Transactions	Yes	No	No	No
SQL	Yes	Yes	Yes	No
Open Source	Yes	Yes	Yes	Yes

Table 4.1: Database solutions comparison

4.1.3 LOAD BALANCER

In order to divide database requests per machine, between the Monitor App and the distributed database system, a Load Balancer was configured. It was chosen HAProxy[2], which in addition to being one of the most popular open-source TCP load-balancers, CockroachDB includes a built-in command for generating automatically the configuration file.

Thus, with load balancing, hardly ever any machine will be overloaded with requests.

4.2 ARCHITECTURE

Figure 4.1 shows the architecture of the Monitor App implementation. It was structured in accordance with the requirements described in Subsection 3.2.1.

Initially, when the Monitor App starts, it connects to the databases, using certificates, in order to use a secure connection, through a load balancer to obtain every information stored there and put them in its cache memory. The load balancer will distribute the requests through all database machines, in order to avoid overloading one with all requests. If the entire database system goes down, the application has the capability to handle the failure. A backup of the last state is stored, in order to start the application even if it cannot access the databases. Also, all information is stored in cache memory to improve response time and to handle failures that occur while the application is running.

Periodically, the databases are updated in the cache memory in order to only connect to external machines that do updates, instead of connecting in every requests. If these

[2] `http://www.haproxy.org`

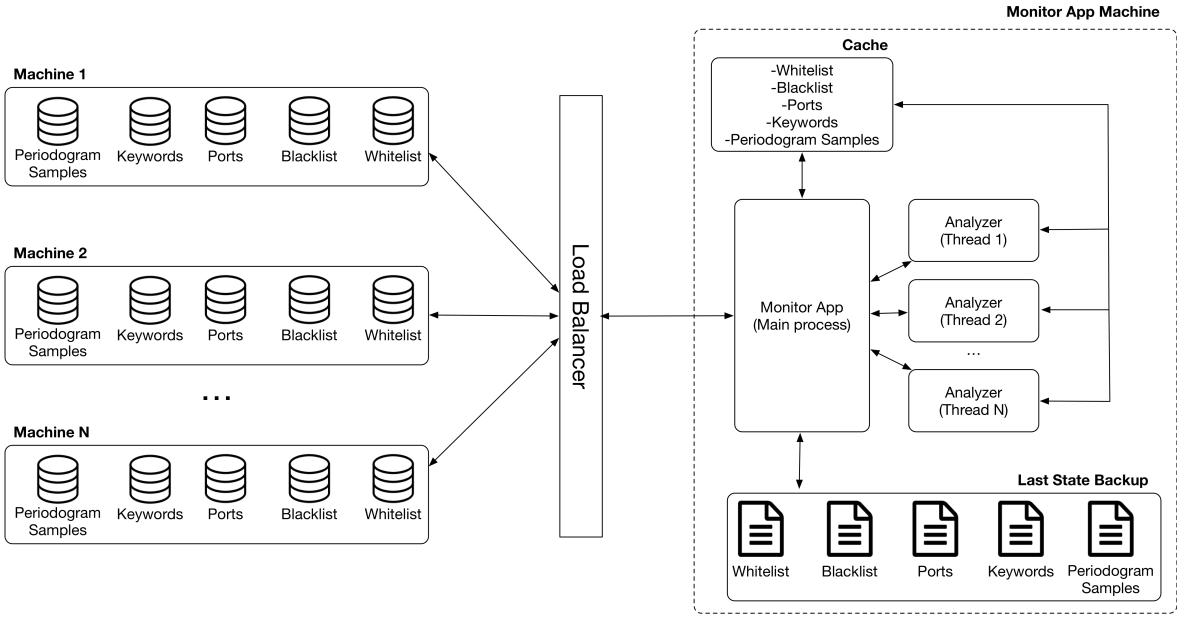

Figure 4.1: Monitor App Implementation: architecture diagram

external machines fail, the system uses what is in the cache memory, handling the failure and continuing its process without access to them.

As previously stated in Subsection 3.2.5, the main process of Monitor App only does immediate analysis, such as consulting the whitelist and the blacklist, and know if the destination network port belongs to a streaming protocol. The most complex analysis are done by the other processes, named analyzers. All these detection mechanisms will be presented in the next section.

4.3 DETECTION MECHANISMS

The main goal of the Monitor App is to detect video streams through network packets flows. In order to achieve this goal, different kinds of analysis are done using multiple information about a certain flow, such as source and destination IPs and ports, hostname assigned to an IP and the frequency of packets from that flow.

In order to understand how these analysis were implemented and how they work, each detection mechanism will be described in the following subsections.

4.3.1 WHITELIST/BLACKLIST

The use of whitelists and blacklists is the easiest way to approve or block anything. In the Monitor App implementation, the whitelist contains all the flows that are video and the blacklist all that are not.

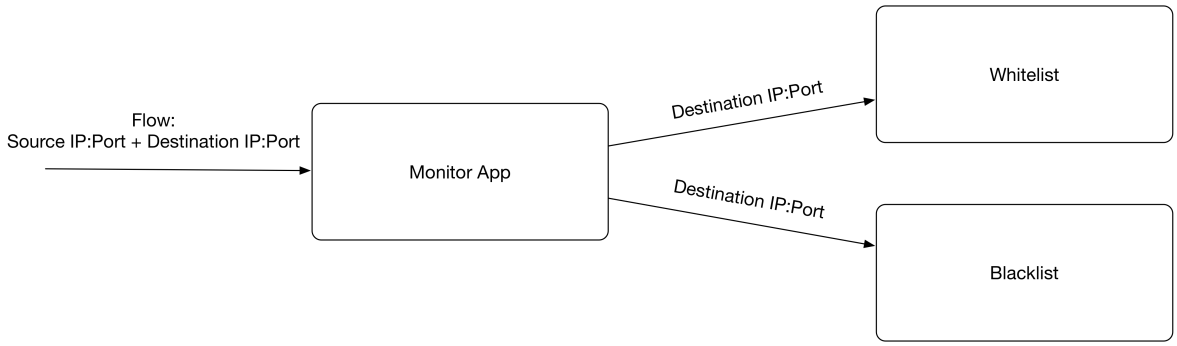

Figure 4.2: Whitelist/Blacklist: Analysis Destination IP:Port

As shown in Figure 4.2, when a flow enters the Monitor App, the destination IP and port are filtered and the application verifies if they are contained in one of the two lists. If the flow is in the whitelist, we know that is a video or if it is in the blacklist, we know that is not. These, are the best cases, because the system can respond immediately. The worst case is if the flow is not in any of the two lists, with the process of detection needing to jump to another mechanism, which is Network Port Analysis, presented in the next subsection.

The data is stored in both databases with the following format:

- x.x.x.x:yyyy, where x.x.x.x corresponds to the destination IP and yyyy corresponds to the destination port of a flow.

4.3.2 NETWORK PORT ANALYSIS

Some multimedia streaming protocols always use the same network port, for example, RTMP uses the TCP port number 1935. Looking at Figure 4.3, it is possible to verify that during a live stream for Youtube, which uses the protocol RTMP, the destination port for the video and audio is always the 1935.

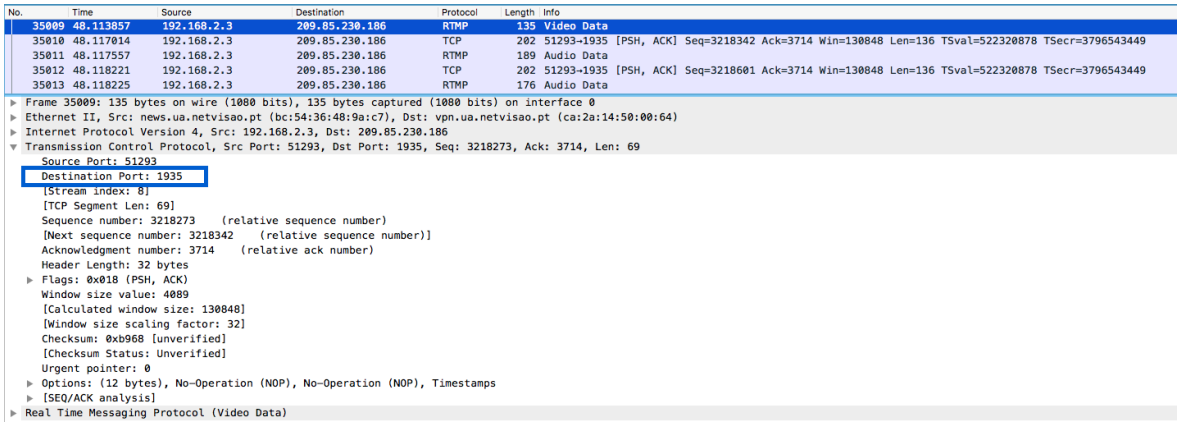

Figure 4.3: Packets RTMP during a Live Stream for Youtube

With this kind of information it is possible to have a list of all ports defined for some protocols.

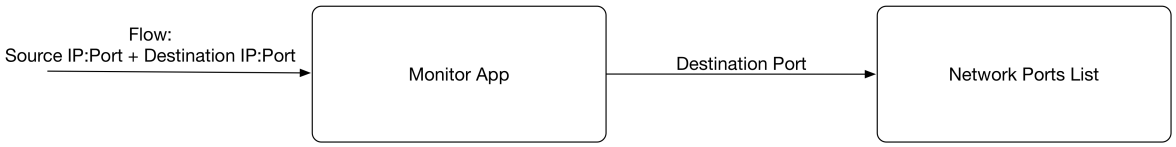

Figure 4.4: Network Port Analysis: Destination Port

Similarly to the Whitelist/Blacklist mechanism, the application filters the flow, using, in this case, the destination port to be analyzed (Figure 4.4). If the port is in the network ports list, the system immediately knows that the flow is video, else, it needs to do another type of analysis, such as the Hostname Analysis shown in next subsection.

4.3.3 HOSTNAME ANALYSIS

Sometimes, through the hostname attributed to an IP, it is possible to understand what is the destination or the purpose of a flow. Through this information, the system can perceive if a flow is video or not, verifying the existence of some keywords in the analyzed hostname.

To obtain that hostname through the IP it is necessary to do a reverse IP lookup, that returns the hostname attributed to it. The system confronts its stored keywords with this information and returns a positive or negative response.

Figure 4.5: Hostname Analysis: Example of a Facebook Stream

Figure 4.5 shows an example of a stream from Facebook. Through the Hostname Analysis, it is possible to find a keyword that gives to the system a partial certainty about the existence of video on that flow, which in this case is "livestream". So, knowing that the destination are the Facebook servers and the hostname contains the keyword "livestream", we can conclude that is a video flow.

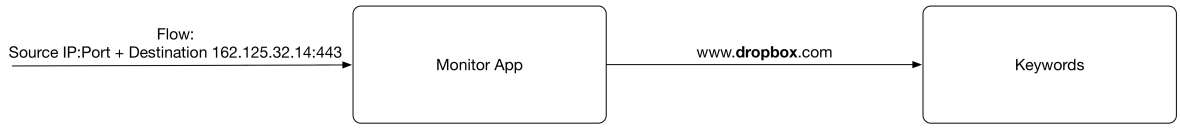

Figure 4.6: Hostname Analysis: Example of a Dropbox access

In order to use this analysis to discard a flow, the keywords can be used as "negative". As an example (Figure 4.6), Dropbox does not do streams of video, so, any flow where the word "dropbox" appears, could be discarded by the system with a non-video response.

If it is impossible to conclude if the flow is video or not, the system uses the last detection mechanism, Periodicity Analysis, which is more complex and slower than all previously presented. The next subsection addresses the description of this mechanism.

4.3.4 PERIODICITY ANALYSIS

Normally, during a flow, the packet arrival frequency generates patterns, regarding to the time between packets. It produces periodical intervals of time, which allow the Periodicity Analysis and posteriorly, the calculation of a periodogram.

A periodogram, in this case is a graphic Time/Power, where the time is in milliseconds (ms) and Power corresponds to the frequency that an interval with χ ms occurs during the packets transmission (Figure 4.7).

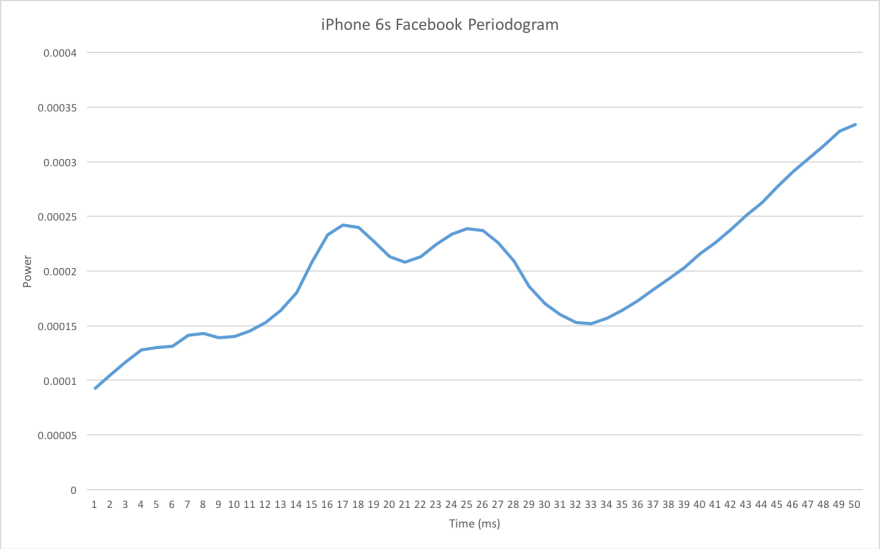

Figure 4.7: Periodicity Analysis: Example of a Periodogram from Facebook Live

Looking at the periodogram of Figure 4.7, we can verify that, normally, intervals between packets last 17 ms and 26 ms, approximately.

To obtain a periodogram, packets are captured during approximately 10 seconds, in order to do a binary analysis, which is an array where the index corresponds to the time in milliseconds and the value inside each index is 1 or 0 depending if any packet arrives in a certain millisecond or not. After this process, a time-frequency representation, named periodogram, is calculated using a Continuous Wavelet Transform (CWT)[37].

Then, the obtained periodogram is compared with samples, used as reference periodograms, in order to obtain the coefficient correlation value, which is a statistical measure of the linear relationship between two variables, to understand if the periodogram corresponds to video or not (Figure 4.8). These samples are a set of 50 values, generated by the CWT and correspond to the frequency of occurrence of an interval of χ ms, from 1 ms to 50 ms. The system returns the best coefficient correlation value of all comparisons, which is the closest to 1. There are three levels of response, Video, Probably Video and Not Video. The first one occurs when the coefficient correlation is comprised between 0.85 and 1, Probably Video occurs when the value is comprised between 0.75 and 0.85 and everything lower than 0.75 is considered Not Video.

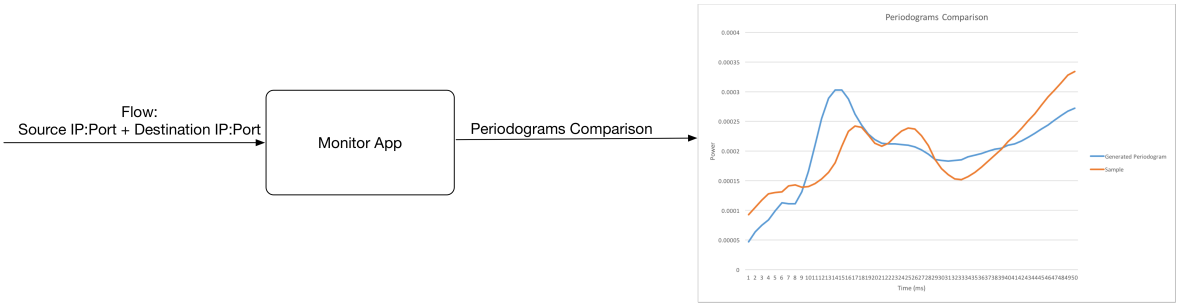

Figure 4.8: Periodicity Analysis: Comparison between two periodograms

As an example of this mechanism, confronting a generated periodogram with a sample (Figure 4.9), we obtain a coefficient correlation of 0.72, which in the Monitor App would be considered Not Video, only comparing with this sample. Normally, there are done multiple comparisons instead of only one.

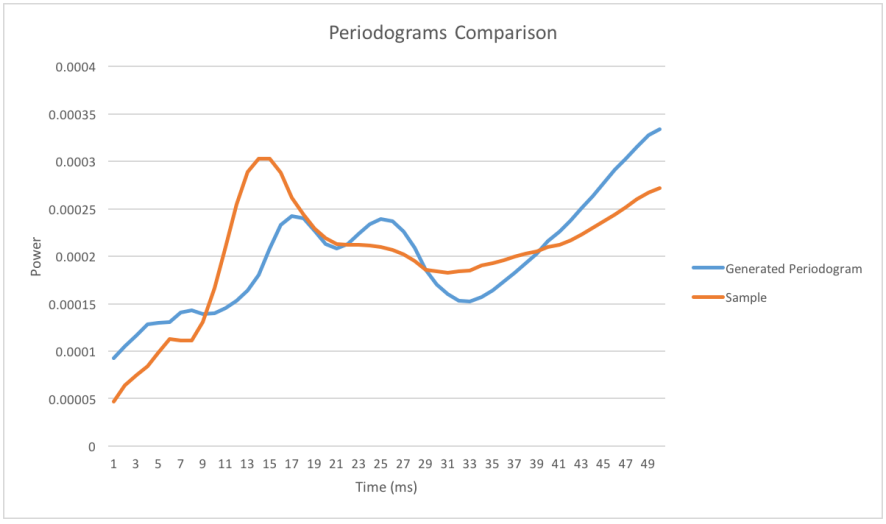

Figure 4.9: Comparison between a generated periodogram and a sample

An analysis of different types of periodograms is presented in Section 5.2 in order to understand the necessity of storing samples obtained by different smartphones using the same platform, for example YouTube, or by the same smartphone using different platforms, such as Facebook and Youtube.

CHAPTER 5

Evaluation and Results

This chapter addresses the deployment of the solution proposed in Chapter 4. Initially, the deployment scenarios are described in 5.1. Then, in 5.2 the patterns generated by multiple video flows in different conditions are analyzed and finally, in 5.3, the latency and efficiency of the system are also analyzed.

5.1 deployment scenarios

As previously described, the solution presented in Chapter 4 can run in a server at the entrance of an ISP or in a BS/AP. In both scenarios, the BS/AP is deployed in an APU1d4 board[1] which includes a built-in Wi-Fi and 3 LAN interfaces. Its CPU is an AMD G series T40E with 1 GHz of clockspeed, two physical cores and it has 4 GB DDR3-1066 of DRAM.

The first scenario (Figure 5.1) addresses a perspective centered at the entrance of a simulated ISP, where a server responsible for running the application exists and all network data flows pass through it. This server was deployed in a Macbook Pro, with 4 GB of memory and a CPU Intel Core i5-2415M with 2.3GHz of clockspeed and two physical cores (2 logical cores per each one).

[1] `http://www.pcengines.ch/apu1d4.htm`

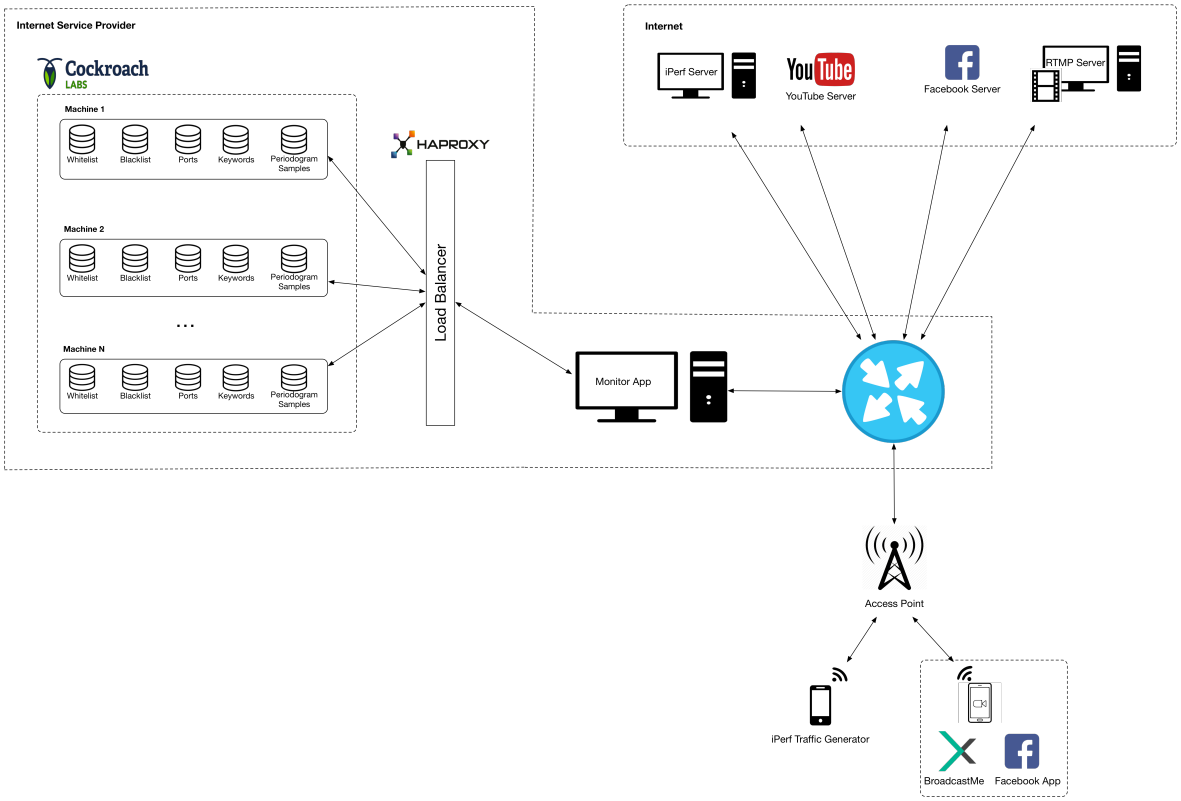

Figure 5.1: Deployment scenario 1 (Monitor App running at the ISP's entrance)

The second one removes the server from the ISP entrance and runs the monitoring application in a BS/AP (Figure 5.2).

In both situations, three different servers were used to do the live streaming, namely YouTube Server, Facebook Server and a RTMP Server running on an Ubuntu Server machine, and an iPerf server which is also running on an Ubuntu Server machine, to where the packets that congest the network are sent. Moreover, three different smartphones (LG E430[38], Nexus 5[39] and iPhone 6s[40]) were used to do the tests presented in Sections 5.2 and 5.3. To congest the network, one of the three smartphones running the Network Tools application[2] was used, doing performance tests with the iPerf Server as destination.

[2] `https://play.google.com/store/apps/details?id=net.he.networktools&hl=en`

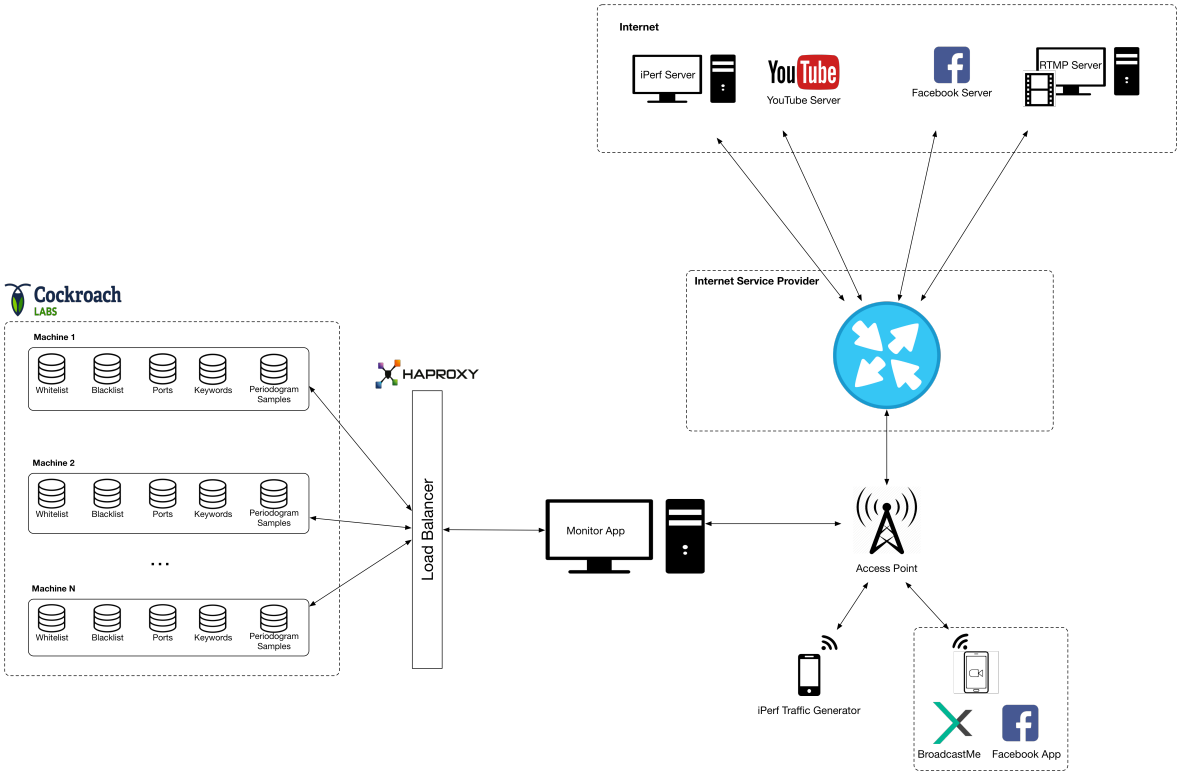

Figure 5.2: Deployment scenario 2 (Monitor App running on a Base Station/Access Point)

5.2 PERIODICITY COMPARISONS

One of the mechanisms of video detection is the Periodicity Analysis, indicated in Section 4.3.4. In order to optimize its efficiency and improve the application, several periodicity comparisons were made. These allowed to know that the periodograms can be different in multiple cases, such as different smartphones, video resolutions and live streaming platforms.

This section analyzes the periodicity comparisons that helped to understand the importance of saving multiple periodograms, in order to prevent the existence of false negatives, when a live stream is not detected.

As previously said, for this analysis three smartphones (LG E430[38], Nexus 5[39] and iPhone 6s[40]) with very distinct characteristics, described in Appendix A, were used and samples with different resolutions and in different situations, such as, face video using the frontal camera, static video, panoramic video or video with movement (Figures 5.3, 5.4, 5.5, 5.6), were captured using all of them. All the samples were

captured using Facebook and BroadcastMe[3] Applications in 640x480, 1280x720 and 1920x1080, except on LG E430, which only allows 640x480, with a 2500 kbps bitrate.

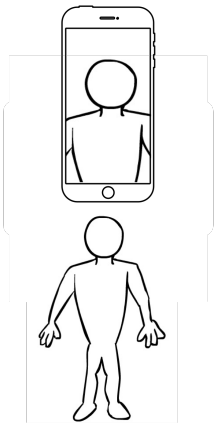

Figure 5.3: Face

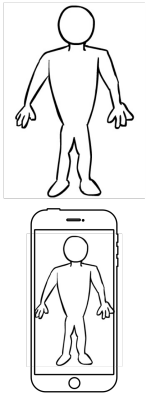

Figure 5.4: Static

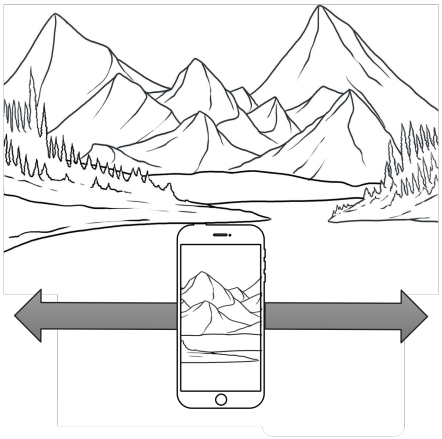

Figure 5.5: Panoramic

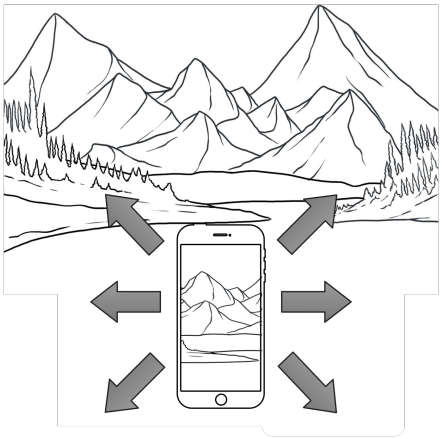

Figure 5.6: Movement

In addition to having used the Facebook and Youtube servers to do the streams, a RTMP server was also used in order to get all the comparisons between smartphones, resolutions and types of video.

5.2.1 SAME SMARTPHONE

5.2.1.1 DIFFERENT VIDEO RESOLUTIONS

Through the periodograms obtained from samples taken by the same smartphone with different resolutions, it is possible to make the comparison of those periodograms.

[3] `https://www.streamaxia.com/broadcastme-whitelabel-app/`

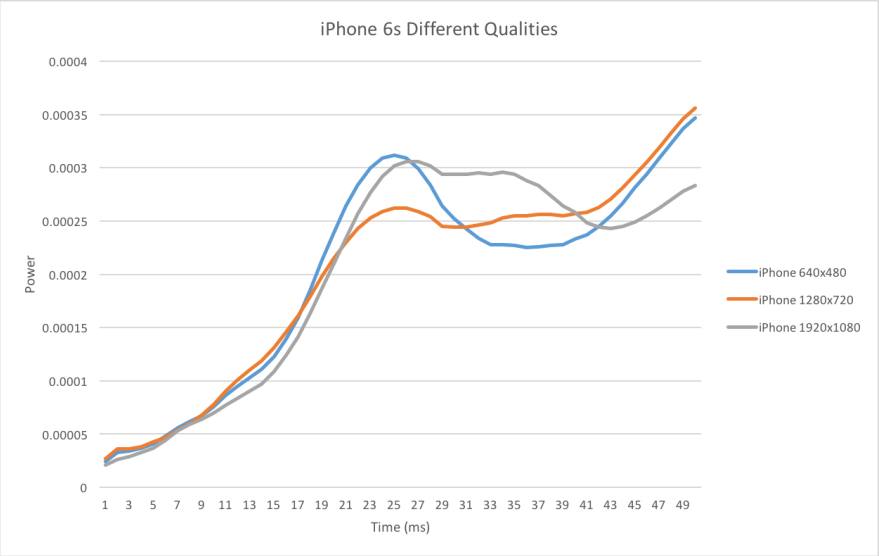

Figure 5.7: iPhone 6s periodograms in different resolutions of video

		Resolutions		
		640x480	**1280x720**	**1920x1080**
Resolutions	**640x480**	1	0,97	0,94
	1280x720	0,97	1	0,94
	1920x1080	0,94	0,94	1

Table 5.1: Correlation coefficient between resolutions (iPhone 6s)

		Resolutions		
		640x480	**1280x720**	**1920x1080**
Resolutions	**640x480**	1	0,58	0,91
	1280x720	0,58	1	0,83
	1920x1080	0,91	0,83	1

Table 5.2: Correlation coefficient between resolutions (Nexus 5)

As only the Nexus 5 and iPhone 6s allow all possible capture resolutions, only the graphics of these two smartphones were analyzed.

In both figures, 5.7 and 5.8, it is possible to understand that small differences exist in the periodograms of the same sample with different resolutions of video and, normally, the correlation coefficient of them is close to 1 (Table 5.1). However, these differences can also generate false information, highlighting the non-existence of a video flow when in reality, that flow exists (Table 5.2 - 640x480 and 1280x720).

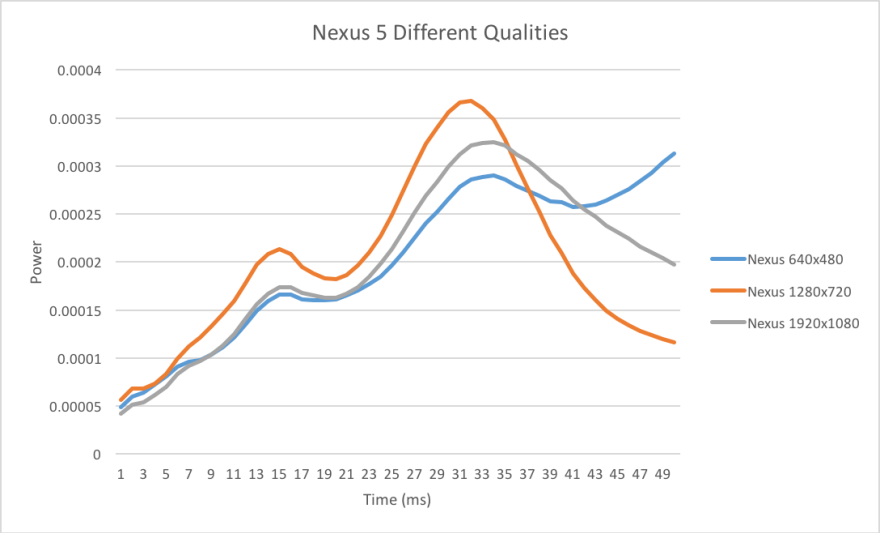

Figure 5.8: Nexus 5 periodograms in different resolutions of video

5.2.1.2 DIFFERENT TYPES OF VIDEO

Also using the same smartphone, but using the same resolution of video in different types of video, the following periodograms were obtained:

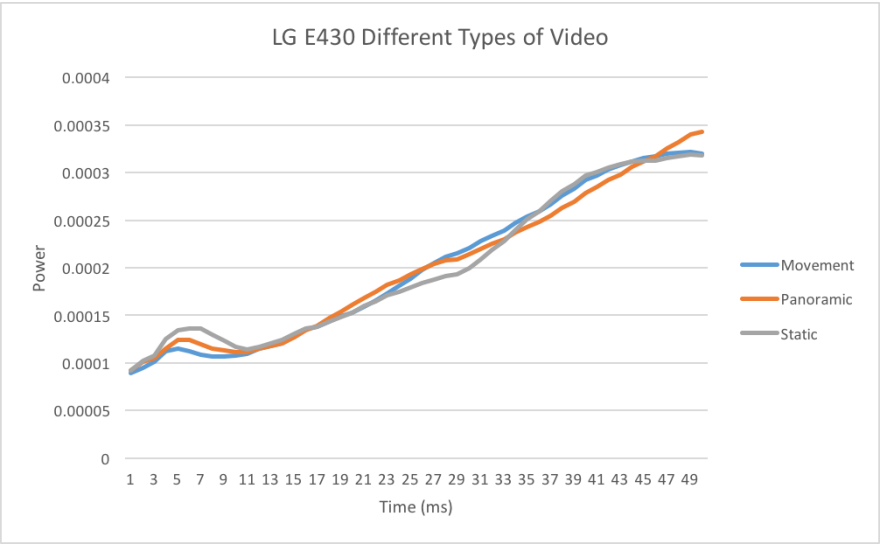

Figure 5.9: LG E430 periodograms in different types of video (640x480)

		Types of video		
		Movement	**Panoramic**	**Static**
Types of video	**Movement**	1	0,99	0,99
	Panoramic	0,99	1	0,99
	Static	0,99	0,99	1

Table 5.3: Correlation coefficient between types of video (LG E430 - 640x480)

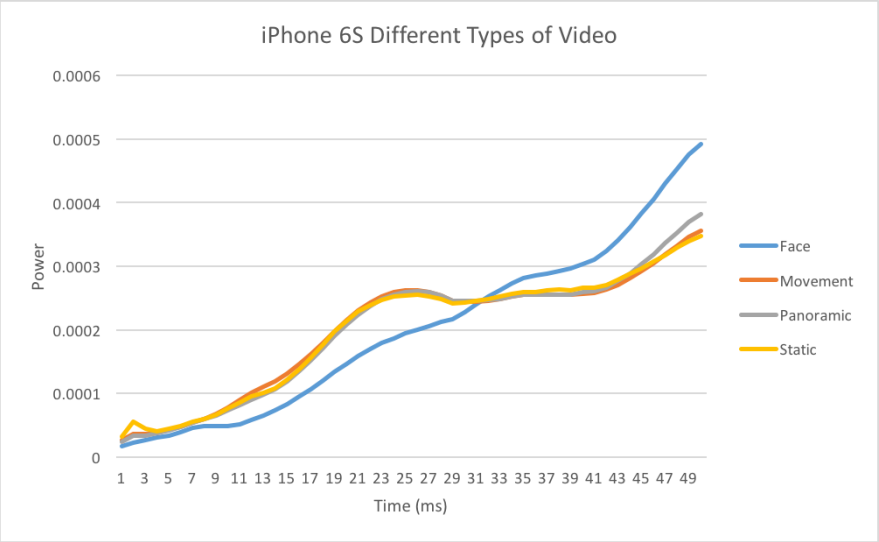

Figure 5.10: iPhone 6s periodograms in different types of video (1280x720)

		Types of video			
		Movement	**Panoramic**	**Static**	**Face**
Types of video	**Movement**	1	1	1	0,92
	Panoramic	1	1	1	0,94
	Static	1	1	1	0,93
	Face	0,92	0,94	0,93	1

Table 5.4: Correlation coefficient between types of video (iPhone 6s - 1280 x 720)

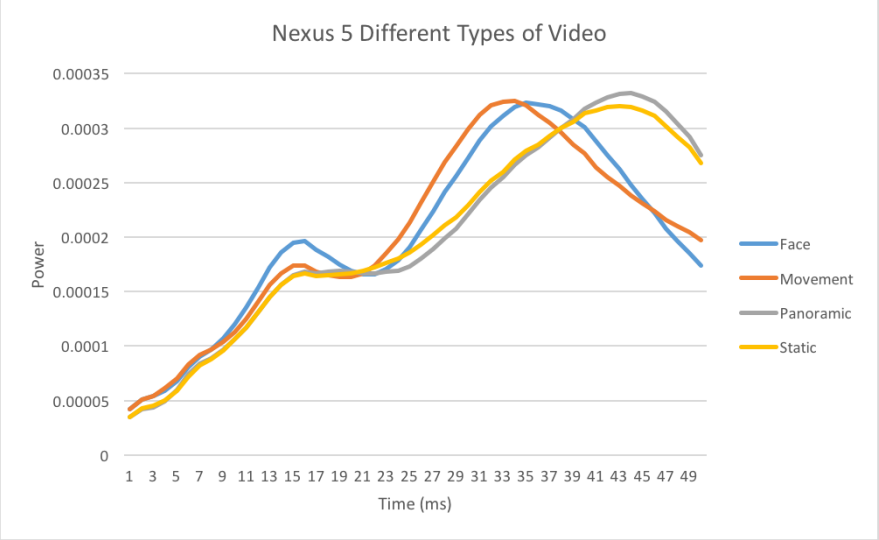

Figure 5.11: Nexus 5 periodograms in different types of video (1920x1080)

Figure 5.9 shows that the type of video does not have any influence in the resulting periodogram, so, between a sample of a static video and a panoramic video the correlation

		Types of video			
		Movement	**Panoramic**	**Static**	**Face**
Types of video	**Movement**	1	0,83	0,87	0,98
	Panoramic	0,83	1	1	0,85
	Static	0,87	1	1	0,88
	Face	0,98	0,85	0,88	1

Table 5.5: Correlation coefficient between types of video (Nexus 5 - 1920 x 1080)

coefficient is approximately 1, and in this case, any video flow would be detected (Table 5.3).

In contrast, Figures 5.10 and 5.11 demonstrate that, depending on the smartphone model, the type of video produces different patterns in the periodograms. Considering the iPhone result, we can infer that differences only exist when using the back camera (Movement, Panoramic, Static) or the frontal camera (Face), which produces a different graphic and it is the only case where the correlation coefficient is less than one. However, it produces a sufficient correlation coefficient to assume the existence of video (Table 5.4). Looking at these results it is not required to save a sample of the frontal camera and other of the back camera. On other hand, the Nexus 5 produces different graphics depending on the type of video and used camera. Using the frontal camera it produces a graphic identical to the one of a video with movement, recorded with the back camera. Complementary to this, a panoramic video and a static video generate a similar pattern. In fact, there are some resemblances in all patterns obtained with the same smartphone. Nevertheless, these similarities are not sufficient to prevent the false negatives which can imply the storage of different patterns for each model (Table 5.5), increasing the number of comparisons needed.

5.2.1.3 YOUTUBE AND FACEBOOK

The last analysis done using the same smartphone was the difference between the patterns produced by Facebook and Youtube streams. Knowing that both platforms use RTMP for live streams, it would be interesting to analyze the generated periodograms. Using the iPhone and Nexus samples of these platforms, the following graphics were obtained:

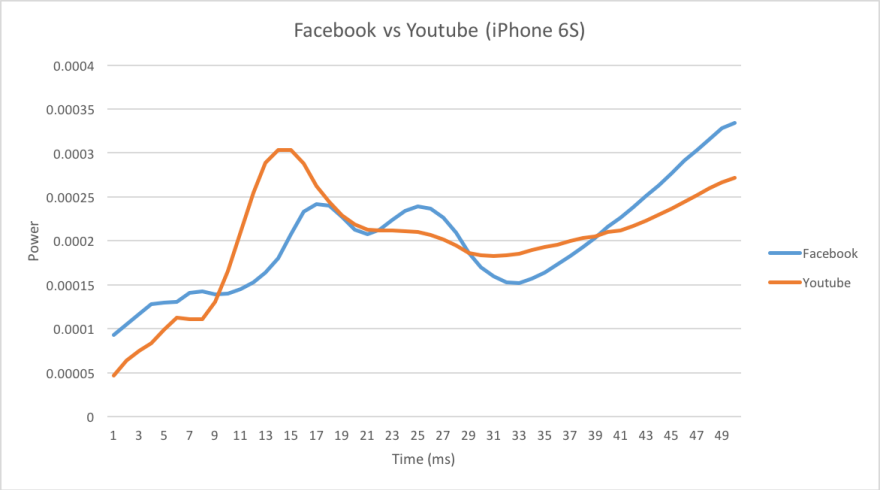

Figure 5.12: Comparison of Facebook and Youtube periodograms (iPhone 6s)

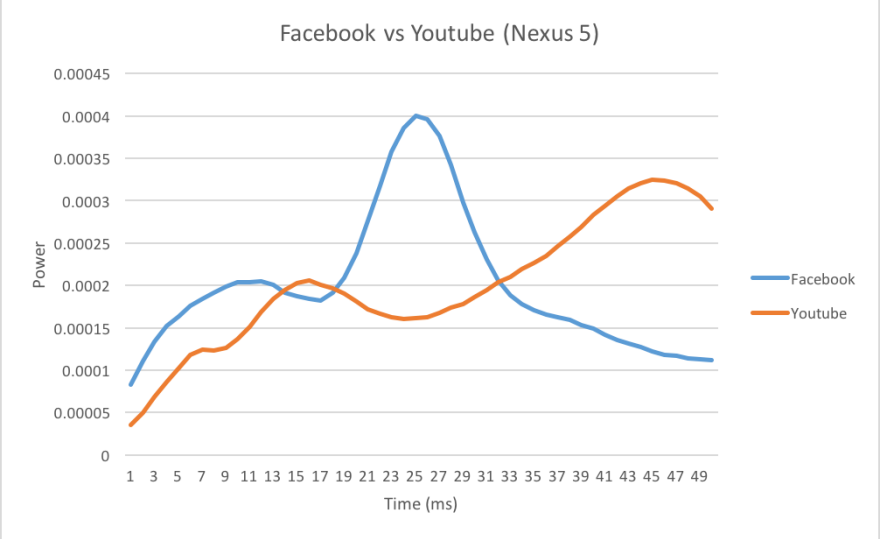

Figure 5.13: Comparison of Facebook and Youtube periodograms (Nexus 5)

Although they use the same streaming protocol, the periodogram of each one is absolutely different in both smartphones and this difference can be derived from the server configurations, where the reduction of latency and the buffering settings are possible causers. The correlation coefficient of the two samples shown in both figures is 0.72 in the iPhone comparison (Figure 5.12) and -0.31 in the Nexus comparison (Figure 5.13), which concludes that the storage of samples of different platforms, such as Facebook and Youtube, is needed.

5.2.2 DIFFERENT SMARTPHONES

5.2.2.1 SAME VIDEO RESOLUTION

Equally important is the study of periodograms to understand the differences when distinct smartphones are used. Firstly, using the same resolution of video and the same type of video, multiple graphics were produced, one per resolution, being that LG E430 only appears in one of them, because it only supports, as previously said, 640x480. Yet, the other resolutions are compared using the Nexus 5 and iPhone 6s.

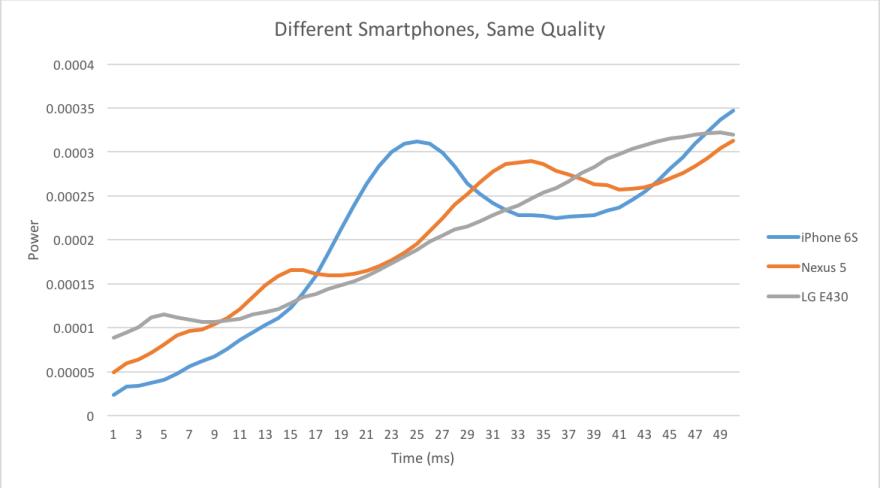

Figure 5.14: Periodograms of different smartphones (640x480)

		Smartphones		
		LG E430	**Nexus 5**	**iPhone 6s**
Smartphones	**LG E430**	1	0,93	0,79
	Nexus 5	0,93	1	0,83
	iPhone 6s	0,79	0,83	1

Table 5.6: Correlation coefficient between different devices (640x480)

Analyzing Figures 5.14, 5.15 and 5.16, we can easily conclude that each smartphone generates its own periodograms. When we analyzed the use of the same smartphone with different resolutions in 5.2.1.1, we conclude that there are small differences between the graphics with a correlation coefficient near to 1, but in this case, different smartphones using the same video resolution produce very distinct patterns. Looking at the results shown in Table 5.6, it is possible to see that Nexus and LG have a high coefficient value but it is not certain that they will always occur. Thus, the storage of periodograms by smartphone model is important. However, devices that produce similar patterns can exist and in those cases it is only necessary to store one corresponding periodogram, saving some comparisons.

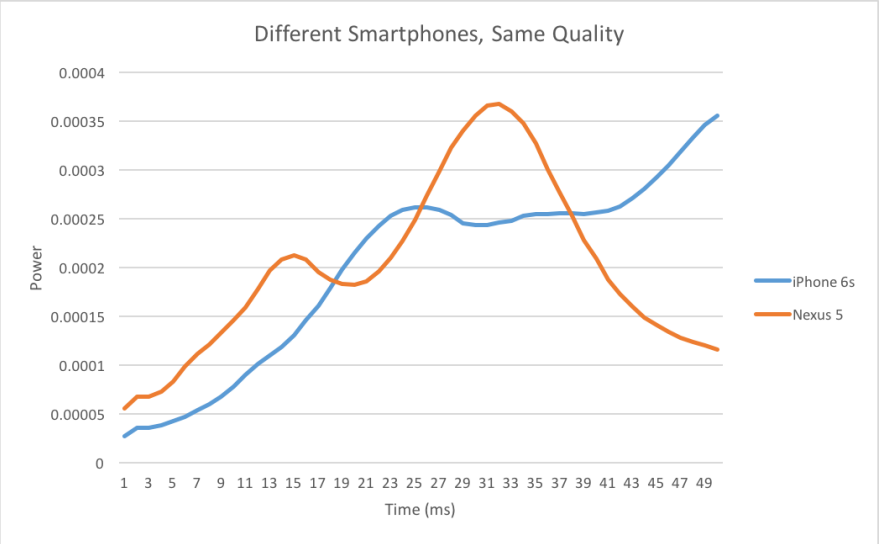

Figure 5.15: Periodograms of different smartphones (1280x720)

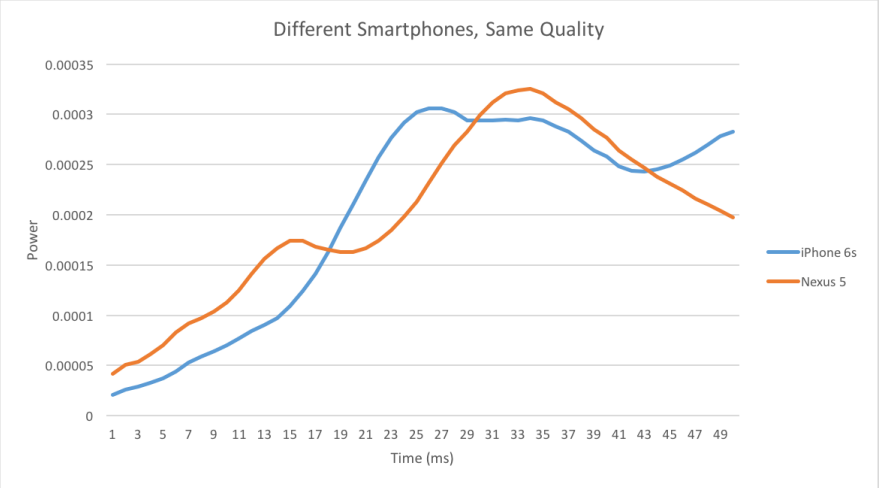

Figure 5.16: Periodograms of different smartphones (1920x1080)

5.2.2.2 DIFFERENT VIDEO RESOLUTIONS

Still using different smartphones, we compared their patterns in distinct resolutions. As using the same device, and distinct resolutions, the patterns have similarities and using different devices and the same resolution the patterns are very different, it is safe to infer that, in this case, the graphics will be also distinct.

As foreseen and shown in Figure 5.17, using as samples, the periodograms of iPhone 6s (1920x1080), Nexus 5 (1280x720) and LG E430 (640x480), the patterns are very distinct which reinforce the idea of storing the samples by device (Table 5.7).

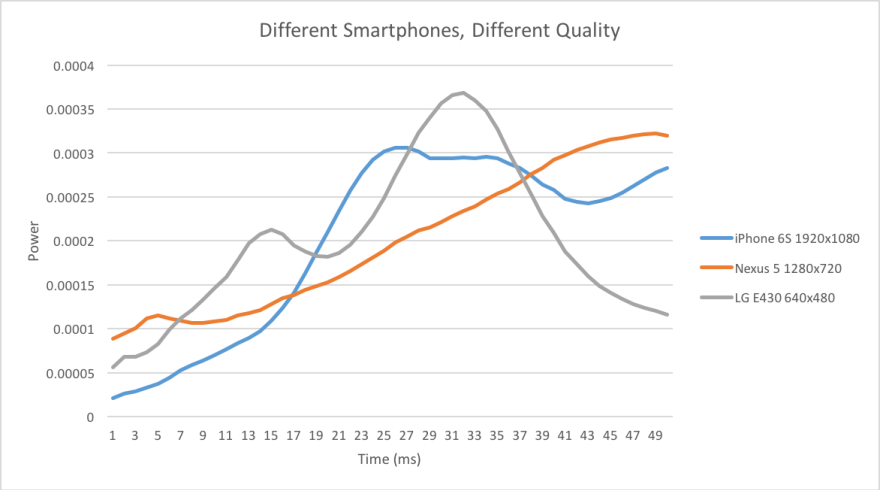

Figure 5.17: Periodograms of different smartphones in distinct resolutions

		Smartphones		
		LG E430	**Nexus 5**	**iPhone 6s**
Smartphones	**LG E430**	1	0,28	0,79
	Nexus 5	0,28	1	0,69
	iPhone 6s	0,79	0,69	1

Table 5.7: Correlation coefficient between different devices and resolutions

5.2.2.3 YOUTUBE AND FACEBOOK

Finally, in order to conclude that each studied device generates a different pattern, samples of Facebook and Youtube streams from the three devices were analyzed and compared. The LG E430 only appears in the Youtube study, because the version of its Facebook application did not support live streams.

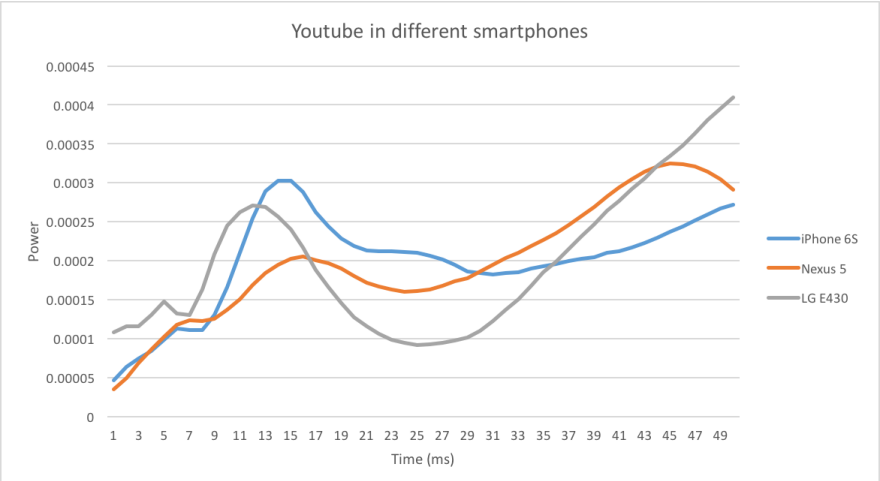

Figure 5.18: Periodograms of different smartphones (Youtube)

		Smartphones		
		LG E430	**Nexus 5**	**iPhone 6s**
Smartphones	**LG E430**	1	0,76	0,54
	Nexus 5	0,76	1	0,69
	iPhone 6s	0,54	0,69	1

Table 5.8: Correlation coefficient of Youtube sample between different devices

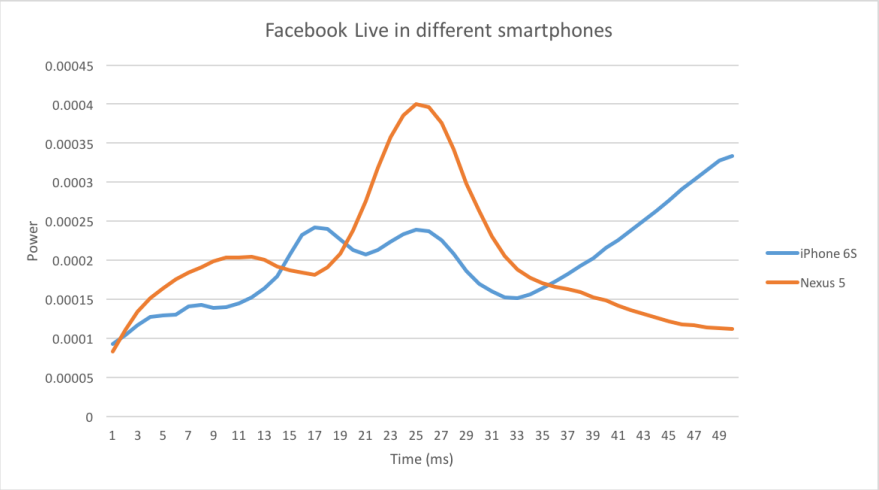

Figure 5.19: Periodograms of different smartphones (Facebook)

Starting with the Youtube analysis (Figure 5.18), the patterns are simultaneously different and similar. If we analyze the peaks, all of them have a small peak between 5ms and 7ms, and a big one between 12ms and 16ms, approximately. But, if we analyze the frequency values, the difference between peaks or the correlation coefficient, all of the graphics are distinct. Looking at the correlation coefficient of these three samples (Table 5.8), we can verify that the samples of LG and Nexus are the more similar with a value of 0.76. In the other cases, the coefficient is 0.54 when LG and iPhone are confronted and 0.69 between Nexus and iPhone.

In the Facebook case (Figure 5.19), analyzing the produced graphic, it is clear that only one peak occurs at approximately the same time, 26ms, and the correlation coefficient is 0. So, if we only store one sample, for example from iPhone, when a Facebook Live stream is done with Nexus, the application would not detect video.

Thus, with these values for both platforms, the storage of samples of each application per device is necessary, but, as previously said different devices can also generate similar patterns, which removes the need of store some samples.

5.3 PERFORMANCE RESULTS

As described in Deployment Scenario 5.1, two different cases were tested. In the first case, the monitoring application is running in a server located after a BS/AP, at the ISP's entrance. The second one removes the server and the monitoring application is running in a BS/AP.

In order to understand the difference between running the application in a server at the ISP's entrance or in a BS/AP, the following sections will analyze the obtained results on the efficiency and latency of the application, using both scenarios in two types of environment, namely, low network usage and overloaded network. To simulate an overloaded network, an iPerf[4] server and a smartphone sending random data to the server were used, in order to congest the network.

5.3.1 SCENARIO 1 - APPLICATION RUNNING IN A SERVER AT THE ISP'S ENTRANCE

5.3.1.1 VIDEO DETECTION LATENCY

As previously stated, initially, the latency of each detection mechanism running the application was obtained in a server located after a BS/AP at the ISP's entrance. It will be responsible for the processing and analysis of the flows, in order to notify the network about the existence of video.

For this scenario, 35 latency measurements were taken and the values correspond to the time that the process took to detect video. These values are shown in Table 5.9 where it is possible to see that, in both environments, namely low network usage and overloaded network, the difference between latencies is small.

		Whitelist/Blacklist	**Network Port**	**Hostname**	**Periodicity**
Low Network Usage	**Average**	0.08 ms	0.11 ms	1.80 ms	12289.14 ms
	Minimum	0.06 ms	0.07 ms	0.98 ms	11023.12 ms
	Maximum	0.11 ms	0.13 ms	5.24 ms	13625.22 ms
	Standard Deviation	0.01 ms	0.01 ms	1.18 ms	535.59 ms
	Confidence Interval	± 0.004 ms	± 0.003 ms	± 0.390 ms	± 180.000 ms
Overloaded Network	**Average**	0.08 ms	0.11 ms	345.14 ms	12780.94 ms
	Minimum	0.07 ms	0.08 ms	1.01 ms	12036.52 ms
	Maximum	0.12 ms	0.15 ms	12020.06 ms	16520.36 ms
	Standard Deviation	0.01 ms	0.01 ms	2411.60 ms	968.10 ms
	Confidence Interval	± 0.004 ms	± 0.003 ms	± 800.000 ms	± 320.000 ms

Table 5.9: Latency of all detection mechanisms (ISP Entrance)

[4] `https://iperf.fr`

The first mechanism is the search of the destination IP and network port in a whitelist or in a blacklist, both stored in a database and in the server's cache. Due to the storage in cache of the whitelist and the blacklist, the system immediately detects the existence of video, and does not need to query the database every time. Thus, there are no differences in both tests of this mechanism. Also, the Network Port Analysis took a similar time to detect the existence of video, because a list of ports, that usually contain video, is also stored in a database and in server's cache.

When the previously mechanisms fails, the application analyzes the hostname associated to the destination IP. This process requires more time to give an answer than the first two, because it is necessary to resolve the hostname and check if it contains keywords, such as, "livestream". Comparing this mechanism in both environments, there is a big difference in the average latency, due to the two video flows that the system took a long time to detect using this manner.

Finally, the Periodicity Analysis was the slowest process, which took approximately 12 seconds to detect video flows in both tests. There is only a big difference in the worst case (maximum latency) of both environments, which was an unique case.

In brief, the latency values do not vary too much when used either in a low network usage environment or in an overloaded network environment, which is an interesting result, taking into account the importance of giving a quick answer to the network. Yet, it is necessary to analyze the efficiency of video detection in both cases, in order to know the possibility of failure.

5.3.1.2 VIDEO DETECTION EFFICIENCY

During the tests, two situations of note were detected. The first one is related with the time that the system took to detect two video flows through the Hostname Analysis mechanism. The system only notified the network after 12 seconds in the worst case, being the normal time approximately 2 milliseconds. These extraordinary cases are the result of an overloaded network, which delays the hostname resolution process.

Another detected efficiency problem occurs when the system cannot obtain a sufficient number of packets to generate a periodogram, due to the quantity of packets that are congesting the network. This problem did not occur frequently, which is a good perspective, taking into account the quantity of flows existent in an overloaded network.

5.3.2 SCENARIO 2 - APPLICATION RUNNING IN A BASE STATION / ACCESS POINT

5.3.2.1 VIDEO DETECTION LATENCY

As this application has multiple scenarios where it can be inserted, it is also necessary to analyze the latency during a video detection when the monitoring tool is running in a BS or in an AP.

In the same way as was done in 5.3.1.1, 35 measurements were taken, being the obtained values the time that the system took to detect the existence of a video flow in two different environments. The results are presented in Table 5.10.

		Whitelist/Blacklist	**Network Port**	**Hostname**	**Periodicity**
Low Network Usage	**Average**	0.17 ms	0.31 ms	2.36 ms	13386.43 ms
	Minimum	0.08 ms	0.11 ms	1.21 ms	12045.23 ms
	Maximum	0.76 ms	0.97 ms	8.54 ms	17423.36 ms
	Standard Deviation	0.16 ms	0.25 ms	1.73 ms	1112.51 ms
	Confidence Interval	± 0.053 ms	± 0.083 ms	± 0.570 ms	± 370.000 ms
Overloaded Network	**Average**	0.27 ms	0.53 ms	383.92 ms	14422.74 ms
	Minimum	0.10 ms	0.12 ms	1.25 ms	12362.51 ms
	Maximum	0.79 ms	1.15 ms	13025.33 ms	27125.63 ms
	Standard Deviation	0.23 ms	0.32 ms	2199.81 ms	2781.60 ms
	Confidence Interval	± 0.076 ms	± 0.110 ms	± 730.000 ms	± 920.000 ms

Table 5.10: Latency of all detection mechanisms (BS/AP)

Analyzing the results, it is possible to conclude that they follow the same pattern of the previously studied latencies (5.3.1.1), where an overloaded network environment generates worse values than a low usage network environment. In general, the values are worse in this scenario comparatively to scenario 1, due to the processing capacity.

Regarding the results obtained, the access to the machine's cache, in the Whitelist/Blacklist and Network Port Analysis, took a similar time in both mechanisms, when used in different environments.

The Hostname Analysis has the same issues previously detected on scenario 1. Sometimes, in an overloaded network environment, it took too much time to detect video, approximately 13 seconds in the worst case, which is an extraordinary situation because the other values are normal and similar to the values of the low usage network environment.

Equally, the system, took similar time in both environments to detect video through the Periodicity Analysis and the worst case took 27 seconds.

5.3.2.2 VIDEO DETECTION EFFICIENCY

As stated in 5.3.1.2, there were only detected two problems of efficiency. The first one concerns to the Hostname Analysis and the other one derives from the Periodicity Analysis. In this scenario the same issues were found, but another one appeared and it is related with the processing, which after a while the application stops the detection of new flows.

The number of flows existent in the network and the lack of capacity to process the threads where the flows are analyzed are the origin of this issue, and it can be solved using a better processor.

In the next section, both scenarios will be compared, in order to understand where the application works better.

5.3.3 SCENARIO 1 AND SCENARIO 2 COMPARISON

Comparing the latency results of both scenarios, we can conclude that only small differences exist due to the processing capacity. In each mechanism the differences are insignificant because they only vary in a few milliseconds except on the Periodicity Analysis that generated differences of 2 seconds maximum, which is a fraction of the overall process length.

Moreover, looking at the efficiency results, in both scenarios it was verified that occasionally, the system, due to an overloaded network, took too much time to detect a video flow using the Hostname Analysis mechanism.

Also, the Periodicity Analysis fails sometimes, when the network is overloaded, because it cannot capture the necessary quantity of packets to generate the flow's periodogram.

There was only one critical point detected that differs the two scenarios, which is the processor of the machines where the application is running. In scenario 2, after a while, due to the processing capacity, the application fails and stops the detection of new flows.

# threads	**Max CPU(%) - Scen. 1**	**Max CPU(%) - Scen. 2**
10	51%	106%
20	74%	190%
50	88%	199%

Table 5.11: Maximum CPU usage, using different number of threads

Looking at Table 5.11, we can conclude that, analyzing the maximum usage of CPU using a different number of threads, scenario 2 has high values which is the cause for the issue previously described, related with the processing capability. Also, Figures 5.20, 5.21 and 5.22, show the percentage of CPU during 50 seconds. In all cases a percentage drop exists, which corresponds to an interval where the number of flows were reduced in order to understand the CPU behavior. It is clear to conclude that in all cases, scenario 2 had worse results, due to the worst processing capacities of the machine that simulates the AP.

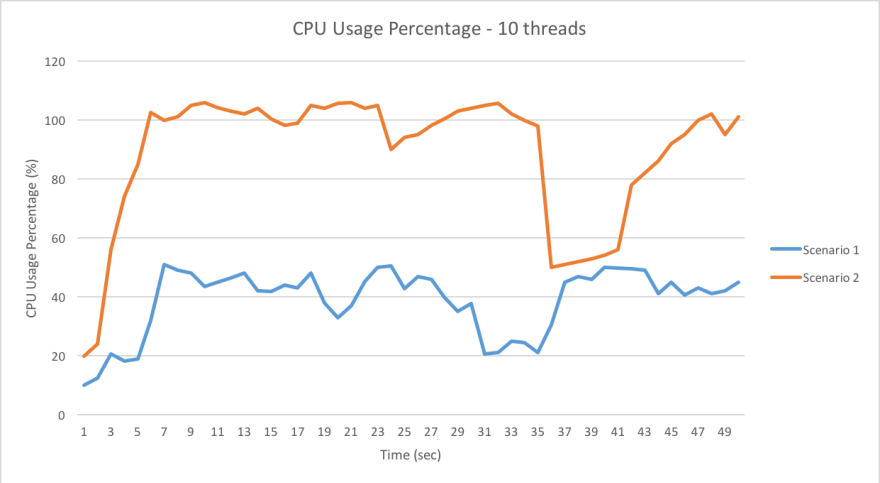

Figure 5.20: CPU Usage Percentage using 10 threads

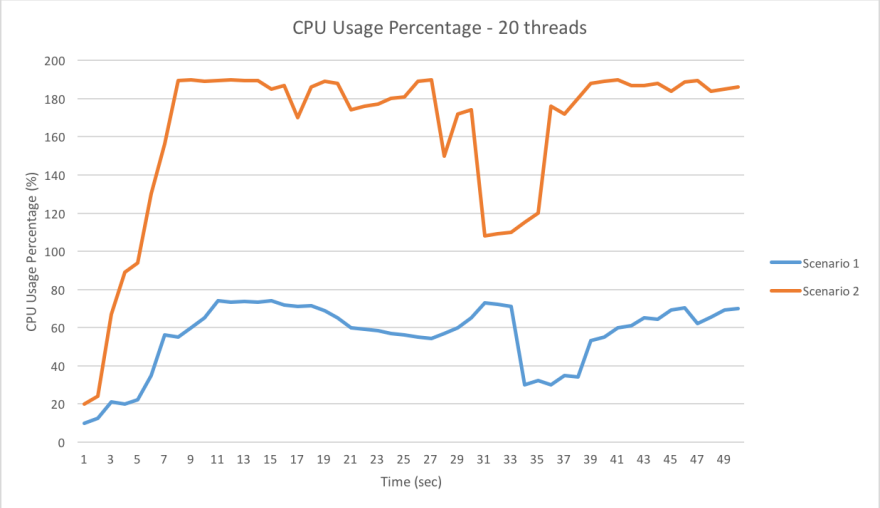

Figure 5.21: CPU Usage Percentage using 20 threads

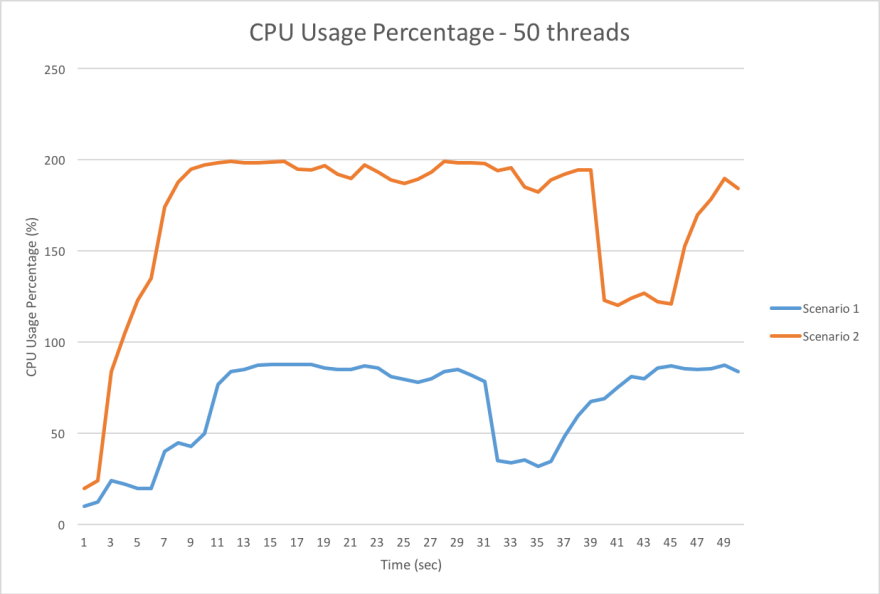

Figure 5.22: CPU Usage Percentage using 50 threads

The necessity of storing periodograms to do the Periodicity Analysis, addressed in 5.2, is responsible for the existence of false positive detections that due to the multiple stored patterns, the probability of a flow be similar to one of them is bigger.

To summarize, there are no big differences between both cases and only the processing capacity distinguishes them, which can be solved by upgrading the processor of the BS/AP.

CHAPTER 6

Conclusion

The increasing appearance of live streaming platforms motivates the creation of new monitoring applications that, together with the QoS improvements enabled by the SDN paradigm, will automate the network, through dynamically optimizations. This automation derives from the separation of control and data planes in SDN networks, which brings a better QoS approach that the traditional network architecture does not support (Section 2.3.3).

This document presents the advantages of a dynamic network defined by software, which is optimized automatically through the use of QoS policies and priorities taking into account the type of traffic. Being video traffic the most relevant to be analyzed, it also presents a solution for a monitoring application that distinguishes all the traffic containing video, through multiple detection mechanisms, for example, the analysis of the periodicity of a certain flow, and notifies about it.

The architecture conceptualized for this implementation, allows the system, where Monitor App is integrated, to receive, as quickly as possible, information about traffic flows. It also proposes a scalable and efficient application, with low percentage of fails and failure handling. Due to the division of different modules, such as the Periodicity Analysis, the architecture presented has two different approaches, either everything running in the same machine or in different machines, which makes this solution scalable.

An overview about how the implementation works is also provided in this document, explaining all the steps of Monitor App, since it receives a network packet until it has informations regarding to the flow of that packet. An explanation of which technologies were adopted and how each detection mechanism was implemented is also presented.

Finally, through the implementation of the presented solution, different results were taken which allows to conclude all the capabilities of the application and define future work, in order to improve found fragilities. These results were analyzed taking into account two different scenarios where the application was tested, running it in different devices, with different network loads.

6.1 FUTURE WORK

Although the provided implementation is a working and tested version of Monitor App, there are some functionalities that were not implemented or were detected with the results analysis. The most important functionalities that should be implemented or improved in the future are:

a) Distributed version

Despite being designed to support a distributed approach using different machines instead of threads running in the same machine, this hypothesis was not implemented, which would be interesting.

b) Improve databases access

Instead of accessing the databases periodically, when databases are updated the Monitor App is notified.

c) Monitoring platform

Implementation of a platform for network managers that uses the Monitor App and provides many types of information, for example, the percentage of video flows, the number of video flows over time, among others.

d) Dynamic Whitelist and Blacklist

Updating both lists dynamically, depending on the percentage of certain flow contain video. For example, if the Monitor App only detects video sporadically in the same flow, we cannot conclude that it is always video, but if the Monitor App constantly detects, we conclude that it is video and the flow is added automatically to the whitelist. The same reasoning is applied to the blacklist, but with non-video information.

e) Improve samples comparison

Due to the large quantity of samples to compare, it would be necessary to do the comparisons in a separated machine, in order to improve the latency of this process.

References

[1] J. Erman and K. K. Ramakrishnan, "Understanding the Super-sized traffic of the Super Bowl", *Proceedings of the ACM SIGCOMM Internet Measurement Conference, IMC*, pp. 353–359, 2013. DOI: 10.1145/2504730.2504770. [Online]. Available: http://www.scopus.com/inward/record.url?eid=2-s2.0-84890020896&partnerID=tZOtx3y1.

[2] Cisco, "Can Software-Defined Networking (SDN) Enhance Operator Monetization ?", pp. 2013–2015, 2014.

[3] M. Syme and P. Goldie, "Understanding Application Layer Protocols", in *Optimizing Network Performance with Content Switching: Server, Firewall and Cache Load Balancing: Server, Firewall, and Cache Load Balancing*, Prentice Hall, 2003, ch. 3.

[4] H. Schulzrinne, C. U., A. Rao, Netscape, R. Lanphier, and RealNetworks, *Real Time Streaming Protocol (RTSP)*, 1998. [Online]. Available: https://www.ietf.org/rfc/rfc2326.txt (visited on 05/09/2017).

[5] H. Parmar and E. M. Thornburgh, "Adobe's Real Time Messaging Protocol", pp. 1–52, 2012.

[6] *Real-Time Media Flow Protocol (RTMFP)*. [Online]. Available: http://www.adobe.com/pt/products/adobe-media-server-extended/rtmfp-faq.html (visited on 05/16/2017).

[7] *Adaptive Streaming*. [Online]. Available: https://bitmovin.com/adaptive-streaming/ (visited on 04/11/2017).

[8] R. N. Yuval Fisher, "An Overview of Http Adaptive Streaming Protocols for Tv Everywhre Delivery", 2014.

[9] *MPEG-DASH*. [Online]. Available: https://www.encoding.com/mpeg-dash/ (visited on 04/12/2017).

[10] I. Sodagar, "MPEG-DASH: The Standard for Multimedia Streaming Over Internet", 2012.

[11] *HTTP Live Streaming Overview*. [Online]. Available: https://developer.apple.com/library/content/documentation/NetworkingInternet/Conceptual/StreamingMediaGuide/Introduction/Introduction.html (visited on 04/12/2017).

[12] *HTTP Streaming Architecture*. [Online]. Available: https://developer.apple.com/library/content/documentation/NetworkingInternet/Conceptual/StreamingMediaGuide/HTTPStreamingArchitecture/HTTPStreamingArchitecture.html#//apple_ref/doc/uid/TP40008332-CH101-SW4 (visited on 04/12/2017).

[13] *Microsoft Smooth Streaming*. [Online]. Available: https://www.encoding.com/microsoft-smooth-streaming/ (visited on 04/13/2017).

[14] A. Zambelli, "IIS Smooth Streaming Technical Overview", *Microsoft Corporation*, no. March, 2009.

[15] *WebRTC*. [Online]. Available: https://webrtc.org/faq/#what-is-webrtc (visited on 05/16/2017).

[16] S. Pietro Romano and S. Loreto, *Real-Time Communication with WebRTC*. O'Reilly Media, Inc., 2014.

[17] IPknowledge, "Traditional vs Software Defined Networking",

[18] Citrix, "SDN 101 : An Introduction to Software Defined Networking", 2014.

[19] Open Networking Foundation, "SDN Architecture Overview", *Onf*, pp. 1–5, 2013.

[20] P. Tanwar, R. Gohil, and M. Tanwar, "Quick Survey of Benefits from Control Plane and Data Plane Separation in Software-Defined Networking", vol. 8, no. 1, pp. 940–943, 2016. DOI: 10.13140/RG.2.1.1605.5442.

[21] *OpenFlow*. [Online]. Available: https://www.opennetworking.org/sdn-resources/openflow (visited on 05/23/2017).

[22] Open Networking Foundation, "OpenFlow Switch Specification", vol. 0, pp. 1–205, 2014.

[23] D. Kreutz, F. M. V. Ramos, P. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-Defined Networking: A Comprehensive Survey", pp. 1–61, 2014. arXiv: 1406.0440. [Online]. Available: http://arxiv.org/abs/1406.0440.

[24] *What Is Open vSwitch*. [Online]. Available: http://docs.openvswitch.org/en/latest/intro/what-is-ovs/ (visited on 05/23/2017).

[25] *Mininet Overview*. [Online]. Available: http://mininet.org/overview/ (visited on 05/23/2017).

[26] S. Tomovic, M. Pejanovic-Djurisic, and I. Radusinovic, "SDN based mobile networks: Concepts and benefits", *Wireless Personal Communications*, vol. 78, no. 3, pp. 1629–1644, 2014, ISSN: 09296212. DOI: 10.1007/s11277-014-1909-6.

[27] H. Nam, D. Calin, and H. Schulzrinne, "Intelligent content delivery over wireless via SDN", *2015 IEEE Wireless Communications and Networking Conference, WCNC 2015*, pp. 2185–2190, 2015, ISSN: 1525-3511. DOI: 10.1109/WCNC.2015.7127806.

[28] R. Haw, C. S. Hong, and S. Lee, "An efficient content delivery framework for SDN based LTE network", *Proceedings of the 8th International Conference on Ubiquitous Information Management and Communication - ICUIMC '14*, pp. 1–6, 2014. DOI: 10.1145/2557977.2558087. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2557977.2558087.

[29] S. Yilmaz, A. M. Tekalp, and B. D. Unluturk, "Video streaming over software defined networks with server load balancing", *2015 International Conference on Computing, Networking and Communications (ICNC)*, pp. 722–726, 2015. DOI: 10.1109/ICCNC.2015.7069435. [Online]. Available: http://ieeexplore.ieee.org/document/7069435/.

[30] D. Raumer, L. Schwaighofer, and G. Carle, "MonSamp: an SDN Application for QoS Monitoring", vol. 2, pp. 961–968, 2014, ISSN: 2300-5963. DOI: 10.15439/2014F175. [Online]. Available: https://fedcsis.org/proceedings/2014/drp/175.html.

[31] *Quality of Service Networking*. [Online]. Available: http://docwiki.cisco.com/wiki/Quality_of_Service_Networking#Basic_QoS_Architecture (visited on 06/05/2017).

[32] R. Parsons, "QoS components: A closer look", PhD thesis. [Online]. Available: http://searchtelecom.techtarget.com/tip/QoS-components-A-closer-look.

[33] *Quality of Service (QoS) in LTE*. [Online]. Available: http://www.simpletechpost.com/2013/01/quality-of-service-qos-in-lte.html (visited on 06/09/2017).

[34] Cisco, *Quality of Service Design Overview*. [Online]. Available: http://www.cisco.com/c/en/us/td/docs/solutions/Enterprise/WAN_and_MAN/QoS_SRND/QoS-SRND-Book/QoSIntro.html (visited on 06/09/2017).

[35] M. Karakus and A. Durresi, "Quality of Service (QoS) in Software Defined Networking (SDN): A survey", *Journal of Network and Computer Applications*, vol. 80, no. December 2016, pp. 200–218, 2017, ISSN: 10958592. DOI: 10.1016/j.jnca.2016.12.019. [Online]. Available: http://dx.doi.org/10.1016/j.jnca.2016.12.019.

[36] *Wireshark*. [Online]. Available: https://www.wireshark.org/ (visited on 06/09/2017).

[37] MathWorks, *Continuous Wavelet Transform and Scale-Based Analysis*. [Online]. Available: https://www.mathworks.com/help/wavelet/gs/continuous-wavelet-transform-and-scale-based-analysis.html (visited on 06/09/2017).

[38] *LG E430*. [Online]. Available: http://www.gsmarena.com/lg_optimus_l3_ii_e430-5292.php (visited on 05/24/2017).

[39] *LG Nexus 5*. [Online]. Available: http://www.gsmarena.com/lg_nexus_5-5705.php (visited on 05/24/2017).

[40] *iPhone 6s*. [Online]. Available: http://www.gsmarena.com/apple_iphone_6s-7242.php (visited on 05/24/2017).

Appendix A: Smartphones Comparison

		LG E430	**Nexus 5**	**Apple iPhone 6s**
Network and WLAN	**Network**	GSM / HSPA	GSM / CDMA / HSPA / LTE	GSM / CDMA / HSPA / EVDO/ LTE
	WLAN	Wi-Fi 802.11 b/g/n	Wi-Fi 802.11 a/b/g/n/ac	Wi-Fi 802.11 a/b/g/n/ac
Display	**Type**	IPS LCS capacitive touchscreen	True HD IPS+ capacitive touchscreen, 16M colors	LED-backlit IPS LCD, capacitive touchscreen, 16M colors
	Reslution	240 x 320 pixels	1080 x 1920 pixels	750 x 1334 pixels
Platform	**OS**	Android 4.1.2 (Jelly Bean)	Android 6.0 (Marshmallow)	iOS 10.3.2
	Chipset	Qualcomm MSM 7225A Snapdragon S1	Qualcomm MSM8974 Snapdragon 800	Apple A9
	CPU	1.0 GHz Cortex A5	Quad-core 2.3 GHz Krait 400	Dual-core 1.84 GHz Twister
Memory	**RAM**	512 MB	2 GB	2 GB
Camera	**Primary**	3.15 MP	8 MP, f/2.4, 30mm	12 MP, f/2.2, 29mm
	Video	480p@30fps	1080p@30fps	2160p@30fps, 1080p@60fps
	Secondary	No	1.3 MP, f/2.4	5 MP, f/2.2, 31mm

Table 1: Smartphones Comparison [38] [39] [40]